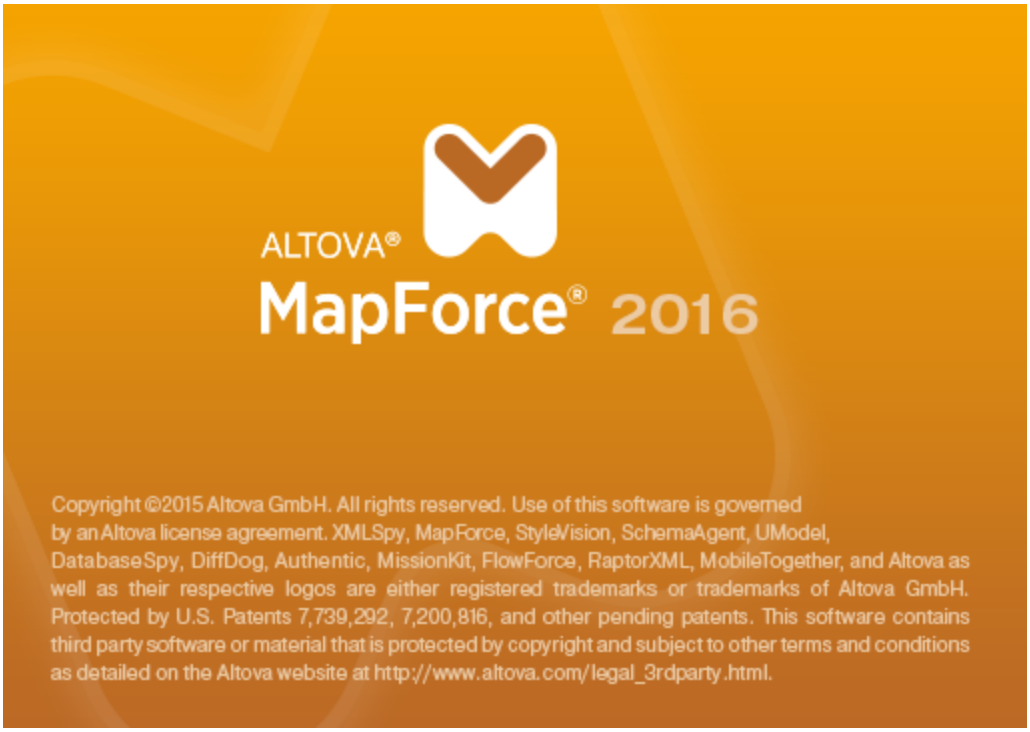
User and Reference Manual


ALTOVA®
MapForce® 2016
Copyright ©2015 Altova GmbH. All rights reserved. Use of this software is governed by an Altova license agreement. XMLSpy, MapForce, StyleVision, SchemaAgent, UModel, DatabaseSpy, DiffDog, Authentic, MissionKit, FlowForce, RaptorXML, MobileTogether, and Altova as well as their respective logos are either registered trademarks or trademarks of Altova GmbH. Protected by U.S. Patents 7,739,292, 7,200,816, and other pending patents. This software contains third party software or material that is protected by copyright and subject to other terms and conditions as detailed on the Altova website at http://www.altova.com/legal_3rdparty.html.

Altova MapForce 2016 User & Reference Manual

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Published: 2015

© 2015 Altova GmbH

Table of Contents

1 MapForce 2016 3

1.1 What's new... 4

2 Introduction 8

2.1 What Is MapForce? 9

2.2 Basic Concepts 15

2.3 User Interface Overview 17

2.4 Conventions 21

3 Tutorials 24

3.1 Convert XML to New Schema 25

3.2 Map Multiple Sources to One Target 36

3.3 Work with Multiple Target Schemas 42

3.4 Process and Generate Files Dynamically 51

4 Common Tasks 62

4.1 Working with Mappings 63

4.1.1 Adding Components to the Mapping 63

4.1.2 Adding Components from a URL 64

4.1.3 Selecting a Transformation Language 67

4.1.4 Validating Mappings 68

4.1.5 Validating the Mapping Output 69

4.1.6 Previewing the Output 70

4.1.7 Previewing the XSLT Code 71

4.1.8 Generating XSLT Code 72

4.1.9 Working with Multiple Mapping Windows 72

4.1.10 Changing the Mapping Settings 74

4.2 Working with Components 76

4.2.1 Searching within Components 77

4.2.2 Aligning Components 78

4.2.3 Changing the Component Settings 79
4.2.4 Duplicating Input 79
4.3 Working with Connections 81
4.3.1 About Mandatory Inputs 82
4.3.2 Changing the Connection Display Preferences 83
4.3.3 Annotating Connections 84
4.3.4 Connection Settings 84
4.3.5 Connection Context Menu 86
4.3.6 Connecting Matching Children 87
4.3.7 Notifications on Missing Parent Connections 89
4.3.8 Moving Connections and Child Connections 90
4.3.9 Keeping Connections After Deleting Components 93
4.3.10 Dealing with Missing Items 95

5 Designing Mappings 100

5.1 Using Relative and Absolute Paths 102
5.1.1 Using Relative Paths on a Component 102
5.1.2 Fixing Broken Path References 104
5.1.3 About Paths in Generated Code 105
5.1.4 Copy-Paste and Relative Paths 106
5.2 Connection Types 107
5.2.1 Target-driven connections 107
5.2.2 Source-driven connections 107
5.2.3 Copy-all connections 115
5.3 Chained Mappings 118
5.3.1 Chained mappings - Pass-through active 119
5.3.2 Chained mappings - Pass-through inactive 124
5.3.3 Chained mapping example 128
5.4 Processing Multiple Input or Output Files Dynamically 131
5.4.1 Mapping Multiple Input Files to a Single Output File 133
5.4.2 Mapping Multiple Input Files to Multiple Output Files 134
5.4.3 Supplying File Names as Mapping Parameters 135
5.4.4 Previewing Multiple Output Files 136
5.4.5 Example: Split One XML File into Many 136
5.5 Supplying Parameters to the Mapping 139
5.5.1 Adding Simple Input Components 139
5.5.2 Simple Input Component Settings 140

5.5.3 Creating a Default Input Value ... 141
5.5.4 Example: Using File Names as Mapping Parameters ... 142
5.6 Returning String Values from a Mapping ... 145
5.6.1 Adding Simple Output Components ... 146
5.6.2 Example: Previewing Function Output ... 146
5.7 Using Variables ... 149
5.7.1 Variables - use cases ... 154
5.8 Sorting Data ... 157
5.9 Filtering Data ... 164
5.10 Using Value-Maps ... 166
5.10.1 Passing data through a Value-Map unchanged ... 169
5.10.2 Value-Map component properties ... 171
5.11 Using If-Else Conditions ... 174
5.12 Mapping Rules and Strategies ... 176
5.12.1 Changing the Processing Order of Mapping Components ... 180

6 Data Sources and Targets 184

6.1 XML and XML schema ... 185
6.1.1 Generating an XML Schema ... 185
6.1.2 XML Component Settings ... 185
6.1.3 Using DTDs as "Schema" Components ... 189
6.1.4 Derived XML Schema Types ... 190
6.1.5 QNames ... 192
6.1.6 Nil Values / Nillable ... 192
6.1.7 Comments and Processing Instructions ... 195
6.1.8 CDATA Sections ... 196
6.1.9 Wildcards - xs:any / xs:anyAttribute ... 197
6.1.10 Mapping to the Root Element ... 202
6.1.11 Merging Data from Multiple Schemas ... 202
6.2 HL7 v3.x to/from XML schema mapping ... 205

7 Functions 208

7.1 Working with Functions ... 209
7.1.1 Priority Context node/item ... 211
7.2 User-defined functions ... 213
7.2.1 Function parameters ... 218

7.2.2 Inline and regular user-defined functions 221
7.2.3 Creating a simple look-up function 222
7.2.4 User-defined function - example 227
7.2.5 Complex user-defined function - XML node as input 231
7.2.6 Complex user-defined function - XML node as output 237
7.2.7 Recursive user-defined mapping 241
7.3 Adding custom XSLT functions 250
7.3.1 Adding custom XSLT 1.0 functions 250
7.3.2 Adding custom XSLT 2.0 functions 254
7.3.3 Aggregate functions - summing nodes in XSLT1 and 2 254
7.4 Regular Expressions 257
7.5 Function Library Reference 260
7.5.1 core | aggregate functions 260
7.5.2 core | conversion functions 265
7.5.3 core | file path functions 272
7.5.4 core | generator functions 274
7.5.5 core | logical functions 277
7.5.6 core | math functions 280
7.5.7 core | node functions 283
7.5.8 core | sequence functions 285
7.5.9 core | string functions 297
7.5.10 xpath2 | accessors 305
7.5.11 xpath2 | anyURI functions 306
7.5.12 xpath2 | boolean functions 307
7.5.13 xpath2 | constructors 307
7.5.14 xpath2 | context functions 308
7.5.15 xpath2 | durations, date and time functions 309
7.5.16 xpath2 | node functions 311
7.5.17 xpath2 | numeric functions 313
7.5.18 xpath2 | qname-related functions 313
7.5.19 xpath2 | string functions 314
7.5.20 xslt | xpath functions 316
7.5.21 xslt | xslt functions 318

8 Automating Mappings and MapForce 322

8.1 Automation with RaptorXML Server 323
8.2 MapForce Command Line Interface 324

9 Customizing MapForce 328

9.1 Altova Global Resources 329
9.1.1 Global Resources - Files 329
9.1.2 Global Resources - Folders 335
9.1.3 Global Resources - Application workflow 337
9.1.4 Global Resources - Properties 345
9.2 Catalog Files 349

10 Menu Reference 356

10.1 File 357
10.2 Edit 360
10.3 Insert 361
10.4 Component 363
10.5 Connection 364
10.6 Function 365
10.7 Output 366
10.8 View 368
10.9 Tools 370
10.10 Window 377
10.11 Help Menu 378
10.11.1 Table of Contents, Index, Search 378
10.11.2 Activation, Order Form, Registration, Updates 378
10.11.3 Other Commands 379

11 Appendices 382

11.1 Engine information 383
11.1.1 XSLT and XQuery Engine Information 383
11.1.2 XSLT and XPath/XQuery Functions 389
11.2 Technical Data 456
11.2.1 OS and Memory Requirements 456
11.2.2 Altova XML Validator 456
11.2.3 Altova XSLT and XQuery Engines 456
11.2.4 Unicode Support 457
11.2.5 Internet Usage 457

11.3 License Information 459
11.3.1 Electronic Software Distribution 459
11.3.2 Software Activation and License Metering 460
11.3.3 Intellectual Property Rights 461
11.3.4 Altova End User License Agreement 461

12 Glossary 476

12.1 C 477
12.2 F 478
12.3 I 479
12.4 M 480
12.5 O 481
12.6 P 482
12.7 S 483
12.8 T 484

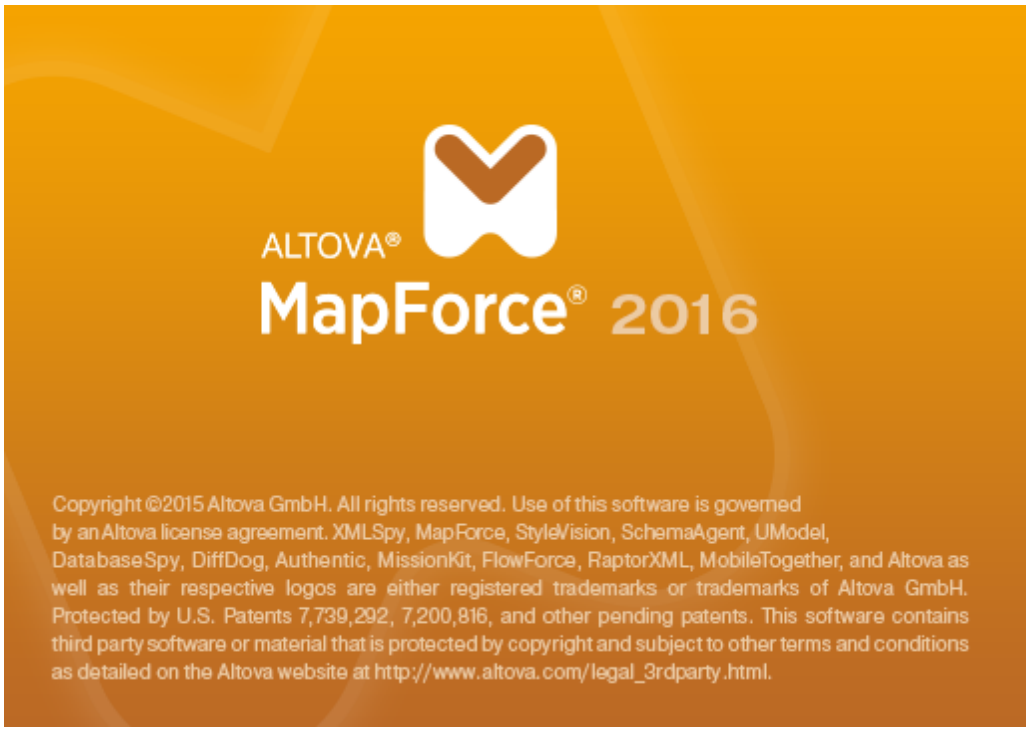
Index

Chapter 1

MapForce 2016

1 MapForce 2016

MapForce® 2016 Basic Edition is a visual data mapping tool for advanced data integration projects. MapForce® is a 32/64-bit Windows application that runs on Windows 10, Windows 8, Windows 7, Windows Vista, Windows XP, and Windows Server 2003/2008/2012. 64-bit support is available for the Enterprise and Professional editions.


ALTOVA®
MapForce® 2016
Copyright ©2015 Altova GmbH. All rights reserved. Use of this software is governed by an Altova license agreement. XMLSpy, MapForce, StyleVision, SchemaAgent, UModel, DatabaseSpy, DiffDog, Authentic, MissionKit, FlowForce, RaptorXML, MobileTogether, and Altova as well as their respective logos are either registered trademarks or trademarks of Altova GmbH. Protected by U.S. Patents 7,739,292, 7,200,816, and other pending patents. This software contains third party software or material that is protected by copyright and subject to other terms and conditions as detailed on the Altova website at http://www.altova.com/legal_3rdparty.html.

Last updated: 24 September 2015

1.1 What's new...

New features in MapForce 2016:

- Improved generation of XSLT 1.0 code (generated stylesheets are easier to read and ofter faster to execute)
- Two new aggregate functions are available in the MapForce core library: **`min-string`** and **`max-string`**. These functions enable you to get the minimum or maximum value from a sequence of strings.

New features in MapForce Version 2015 R4:

- Internal updates and optimizations

New features in MapForce Version 2015 R3 include:

- Option to suppress the `<?xml ... ?>` declaration in XML output
- New component type: Simple Output
- Internal updates and optimizations

New features in MapForce Version 2015 include:

- New `language` argument available in the format-date and format-dateTime functions
- New sequence function: replicate-item

New features in MapForce Version 2014 R2 include:

- New sequence functions: generate sequence, item-at, etc.
- Ability to define CDATA sections in output components
- Keeping connections after deleting components
- Automatic highlighting of mandatory items in target components

New features in MapForce Version 2014 include:

- Integration of RaptorXML validator and basic support for XML Schema 1.1
- Integration of new RaptorXML XSLT engines
- XML Schema Wildcard support, xs:any and xs:anyAttribute
- Support for Comments and Processing Instructions in XML target components

New features in MapForce Version 2013 R2 SP1 include:

- New super-fast transformation engine

New features in MapForce Version 2013 R2 include:

© 2015 Altova GmbH

- Internal updates and optimizations.

New features in MapForce Version 2013 include:

- Internal updates and optimizations

New features in MapForce Version 2012 R2 include:

- New Sort component for XSLT 2.0, XQuery, and the Built-in execution engine
- User defined component names

New features in MapForce Version 2012 include:

- Auto-alignment of components in the mapping window
- Prompt to connect to target parent node
- Specific rules governing the sequence that components are processed in a mapping

New features in MapForce Version 2011R3 include:

- Intermediate variables

New features in MapForce Version 2011R2 include:

- Find function capability in Library window
- Reverse mapping
- Extendable IF-ELSE function
- Node Name and parsing functions in Core Library

New features in MapForce Version 2011 include:

- Ability to preview intermediate components in a mapping chain of two or more components connected to a target component (pass-through preview).
- Formatting functions for dateTime and numbers for all supported languages
- Enhancement to auto-number function

New features in MapForce Version 2010 Release 3 include:

- Support for Nillable values, and xsi:nil attribute in XML instance files
- Ability to disable automatic casting to target types in XML documents

New features in MapForce Version 2010 Release 2 include:

- Automatic connection of identical child connections when moving a parent connection
- Ability to tokenize input strings for further processing

New features in MapForce Version 2010 include:

- Multiple input/output files per component
- Upgraded relative path support
- xsi:type support allowing use of derived types
- New internal data type system
- Improved user-defined function navigation
- Enhanced handling of mixed content in XML elements

New features in MapForce Version 2009 SP1 include:

- Parameter order in user-defined functions can be user-defined
- Ability to process XML files that are not valid against XML Schema
- Regular (Standard) user-defined functions now support complex hierarchical parameters

New features in MapForce Version 2009 include:

- EDI HL7 versions 3.x XML as source and target components
- Grouping of nodes or node content
- Ability to filter data based on a nodes position in a sequence
- QName support
- Item/node search in components

New features in MapForce Version 2008 Release 2 include:

- Ability to automatically generate XML Schemas for XML files
- Support for Altova Global Resources
- Performance optimizations

New features in MapForce Version 2008 include:

- Aggregate functions
- Value-Map lookup component
- Enhanced XML output options: pretty print XML output, omit XML schema reference and Encoding settings for individual components
- Various internal updates

© 2015 Altova GmbH

Chapter 2

Introduction

2 Introduction

This introduction includes an overview of the MapForce features and user interface, the basic concepts in MapForce, as well as the conventions used in this documentation.

© 2015 Altova GmbH

2.1 What Is MapForce?

MapForce is a Windows-based, multi-purpose IDE (integrated development environment) that enables you to transform data from one format to another, or from one schema to another, by means of a visual, "drag-and-drop" -style graphical user interface that does not require writing any program code. In fact, MapForce generates for you the program code which performs the actual data transformation (or data mapping). When you prefer not to generate program code, you can just run the transformation using the MapForce built-in transformation language (available in the MapForce Professional or Enterprise Editions).

You can use MapForce to conveniently convert and transform data from and to a variety of file-based and other formats. Regardless of the technology you work with, MapForce determines automatically the structure of your data, or gives you the option to supply a schema for your data, or generate it automatically from a sample instance file. For example, if you have an XML instance file but no schema definition, MapForce can generate it for you, thus making the data inside the XML file available for mapping to other files or formats.

The technologies supported as mapping sources or targets are as follows.

MapForce Basic Edition	MapForce Professional Edition	MapForce Enterprise Edition
• XML and XML schema • HL7 version 3.x (schema-based)	• XML and XML schema • Flat files, including comma-separated values (CSV) and fixed-length field (FLF) format • Databases (all major relational databases, including Microsoft Access and SQLite databases)	• XML and XML schema • Flat files, including comma-separated values (CSV) and fixed-length field (FLF) format • Data from legacy text files can be mapped and converted to other formats with MapForce FlexText • Databases (all major relational databases, including Microsoft Access and SQLite databases) • EDI family of formats (including UN/EDIFACT, ANSI X12, HL7, IATA PADIS, SAP IDoc, TRADACOMS) • JSON files • Microsoft Excel 2007 and later files • XBRL instance files and taxonomies

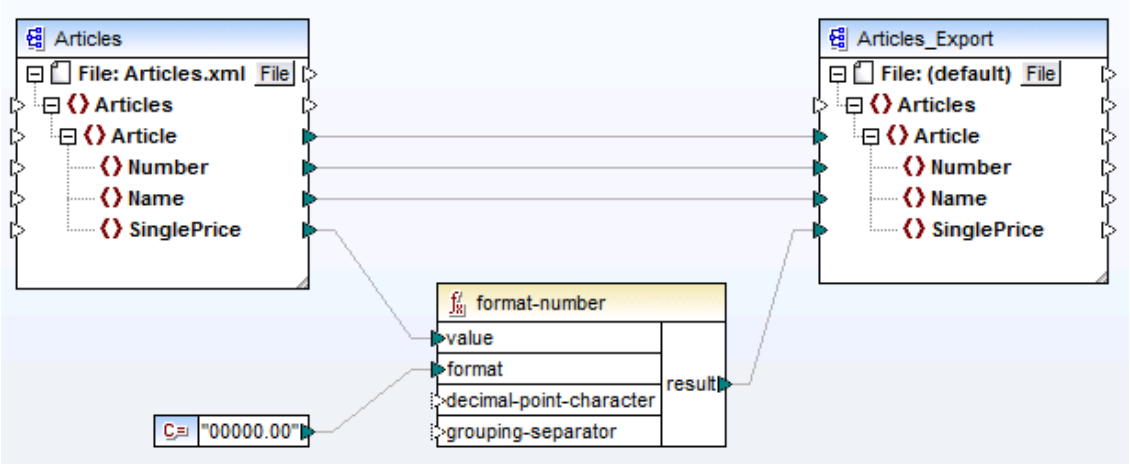
Based on the MapForce edition, you can choose the preferred language for your data transformation as follows.

MapForce Basic Edition	MapForce Professional Edition	MapForce Enterprise Edition
• XSLT 1.0 • XSLT 2.0	• MapForce built-In transformation language • XSLT 1.0 • XSLT 2.0 • XQuery • Java • C# • C++	• MapForce built-In transformation language • XSLT 1.0 • XSLT 2.0 • XQuery • Java • C# • C++

You can preview the result of all transformations, as well as the generated XSLT or XQuery code without leaving the graphical user interface. Note that, as you design or preview mappings, MapForce validates the integrity of your schemas or transformations and displays any validation errors in a dedicated window, so that you can immediately review and address them.

When you choose Java, C#, or C++ as transformation language, MapForce generates the required projects and solutions so that you can open them directly in Visual Studio or Eclipse, and run the generated data mapping program. For advanced data integration scenarios, you can also extend the generated program with your own code, using Altova libraries and the MapForce API.

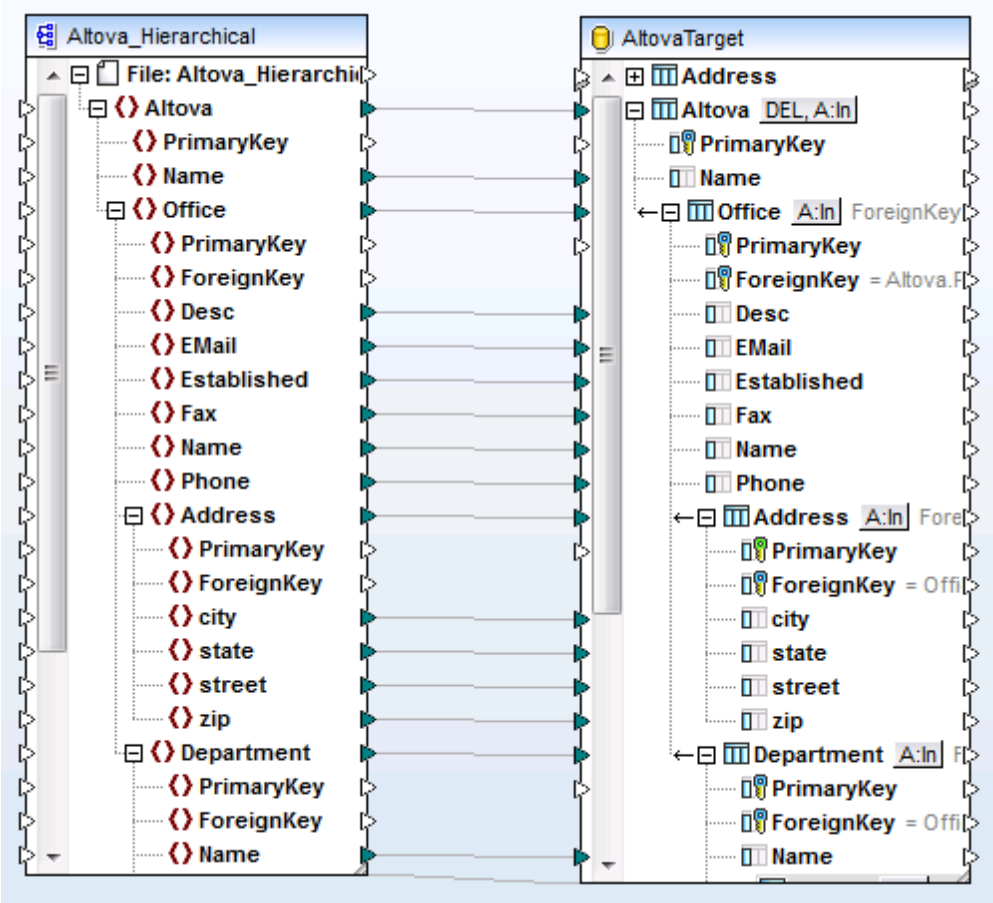
In MapForce, you design all mapping transformations visually. For example, in case of XML, you can connect any element, attribute, or comment in an XML file to an element or attribute of another XML file, thus instructing MapForce to read data from the source element (or attribute), and write it to the target element (or attribute).



Sample data transformation between two XML files

Likewise, when working with databases in MapForce Professional or Enterprise Editions, you can see any database column in the MapForce mapping area and map data to or from it by making visual connections. As with other Altova MissionKit products, when setting up a database connection from MapForce, you can flexibly choose the database driver and the connection type (ADO, ODBC, or JDBC) according to your existing infrastructure and data mapping needs. Additionally, you can visually build SQL queries, use stored procedures, or query a database directly (support varies by database type, edition and driver).

 © 2015 Altova GmbH



Sample data transformation between an XML file and a database

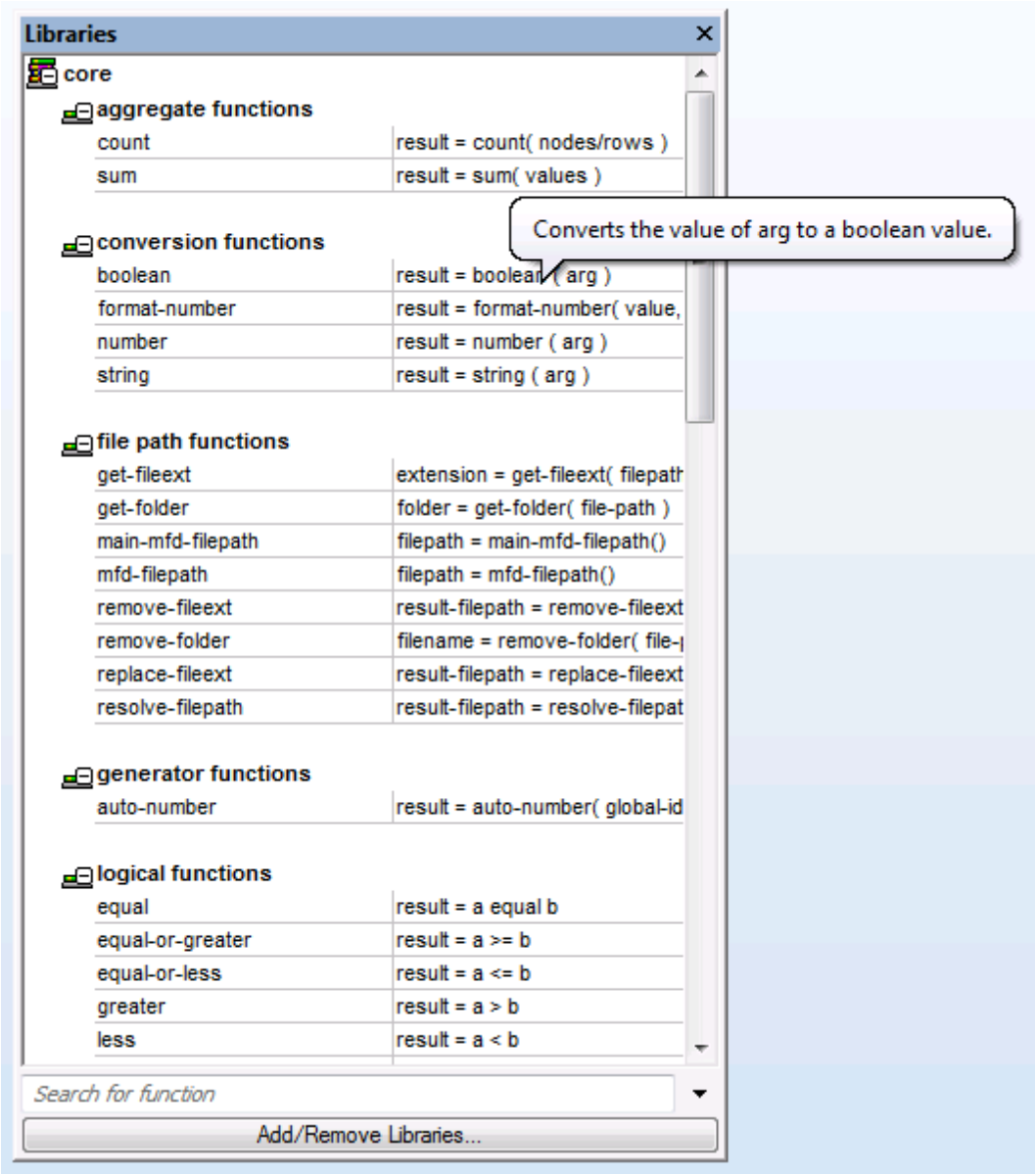
In a very simple scenario, a mapping design created with MapForce could be resumed as "read data from the source X and write it to target Y". However, you can easily design MapForce scenarios such as "read data from the source X and write it to target Y, and then read data from the source Y and write it to the target Z". These are known as "pass-through", or "chained" mappings, and enable you to access your data at an intermediary stage in the transformation process (in order to save it to a file, for example).

Note that the data mappings you can create in MapForce are not limited to single, predefined files. In the same transformation, you can process dynamically multiple input files from a directory and generate multiple output files. Therefore, you can have scenarios such as "read data from multiple X files and write it to a single Y file", or "read file X and generate multiple files Y", and so on.

Importantly, in the same transformation, you can mix multiple sources and multiple targets, which can be of any type supported by your MapForce edition. For example, in case of MapForce Professional or Enterprise, this makes it possible to merge data from two different databases into a single XML file. Or, you can merge data from multiple XML files, and write some of the data to one database, and some of the data to another database. You can preview the SQL statements before committing them to the database.

Direct conversion of data from a source to a target is not typically the only thing you want to achieve. In many cases, you might want to process your data in a particular way (for example, sort, group or filter it) before it reaches the destination. For this reason, MapForce includes, on one hand, miscellaneous functional components that are simplified programming language constructs (such as constants, variables, SQL-WHERE conditions, Filter and Sort components). On the other hand, MapForce includes rich and extensible function libraries which can assist you with virtually any kind of data manipulation.

If necessary, you can extend the built-in library either with functions you design in MapForce directly (the so-called User-Defined Functions, or UDF), or with functions or libraries created externally in XSLT, XQuery, Java, or C# languages.



Libraries pane (MapForce Basic Edition)

 © 2015 Altova GmbH

When your data mapping design files become too many, you can organize them into mapping projects (available in MapForce Professional and Enterprise edition). This allows for easier access and management. Importantly, you can generate program code from entire projects, in addition to generating code for individual mappings within the project.

For advanced data processing needs (such as when running mapping transformations with the MapForce Server API), you can design a mapping so that you can pass values to it at run-time, or get a simple string value from it at run-time. This feature also enables you to quickly test the output of functions or entire mappings that produce a simple string value. The Professional and Enterprise editions of MapForce also include components that enable you to perform run-time string parsing and serialization, similar to how this works in many other programming languages.

With MapForce Enterprise Edition, you can visually design SOAP 1.0 and SOAP 2.0 Web services based on Web Service Language Definition (WSDL) files. You can also call and get data from a WSDL 1.0 or a WSDL 2.0 Web service from within a mapping. This includes Web services available both through the HTTP and HTTPS protocols, as well as Web services which require that the caller uses the WS-Security mechanism, or HTTP authentication.

With MapForce Professional and Enterprise Editions, you can generate detailed documentation of your mapping design files, in HTML, Word 2007+, or RTF formats. Documentation design can be customized (for example, you can choose to include or exclude specific components from the documentation).

If you are using MapForce alongside other Altova MissionKit products, MapForce integrates with them as well as with the Altova server-based products, as shown in the following table.

<table>
<tr><th>MapForce Basic Edition</th><th>MapForce Professional Edition</th><th>MapForce Enterprise Edition</th></tr>
<tr><td colspan="3">You can choose to run the generated XSLT directly in MapForce and preview the data transformation result immediately. When you need increased performance, you can process the mapping using RaptorXML Server, an ultra-fast XML transformation engine.</td></tr>
<tr><td colspan="3">If XMLSpy is installed on the same machine, you can conveniently open and edit any supported file types, by opening XMLSpy directly from the relevant MapForce contexts (for example, the Component | Edit Schema Definition in XMLSpy menu command is available when you click an XML component).</td></tr>
<tr><td colspan="2"></td><td>You can run data transformations either directly in MapForce, or deploy them to a different machine and even operating system for command-line or automated execution. More specifically, you can design mappings on Windows, and run them on a Windows, Linux, or Mac server machine which runs MapForce Server or FlowForce Server.</td></tr>
<tr><td colspan="2"></td><td>If StyleVision is installed on the same machine, you can design or reuse existing StyleVision Power Stylesheets and preview the result of the mapping transformations as HTML, RTF, PDF, or Word 2007+ documents.</td></tr>
</table>

MapForce Professional and Enterprise edition can be installed as a plug-in of Visual Studio and Eclipse integrated development environments. This way, you can design mappings and get access to MapForce functionality without leaving your preferred development environment.

In MapForce, you can completely customize not only the look and feel of the development environment (graphical user interface), but also various other settings pertaining to each technology and to each mapping component type, for example:

- When mapping to or from XML, you can choose whether to include a schema reference, or whether the XML declaration must be suppressed in the output XML files. You can also choose the encoding of the generated files (for example, UTF-8).
- When mapping to or from databases, you can define settings such as the time-out period for executing database statements, whether MapForce should use database transactions, or whether it should strip the database schema name from table names when generating code.
- In case of XBRL, you can select the structure views MapForce should display (such as the "Presentation and definition linkbases" view, the "Table Linkbase" View, or the "All concepts" view).

All editions of MapForce are available as a 32-bit application. The MapForce Professional and Enterprise editions are additionally available as a 64-bit application.

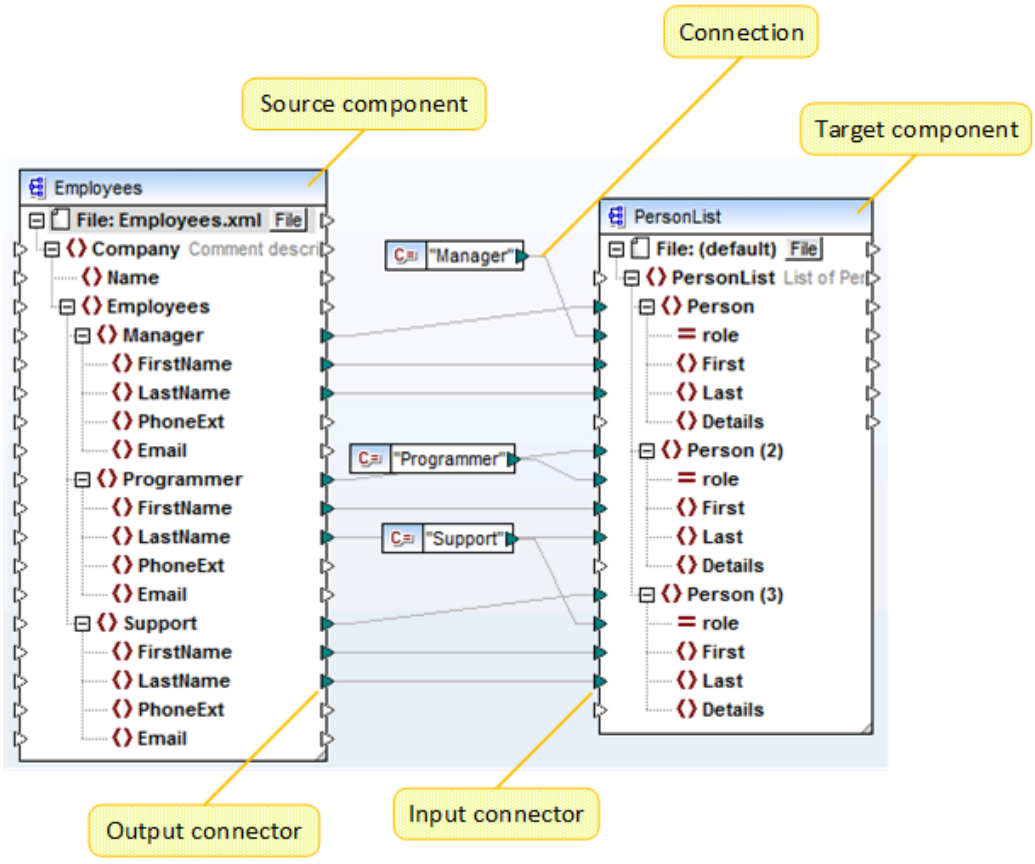
© 2015 Altova GmbH

2.2 Basic Concepts

This section outlines the basic concepts that will help you get started with data mapping.

Mapping

A MapForce mapping design (or simply "mapping") is the visual representation of how data is to be transformed from one format to another. A mapping consists of components that you add to the MapForce mapping area in order to create your data transformations (for example, convert XML documents from one schema to another). A valid mapping consists of one or several source components connected to one or several target components. You can run a mapping and preview its result directly in MapForce. You can generate code and execute it externally. You can also compile a mapping to a MapForce execution file and automate mapping execution using MapForce Server or FlowForce Server. MapForce saves mappings as files with .mfd extension.



Basic structure of a MapForce mapping

Component

In MapForce, the term "component" is what represents visually the structure (schema) of your data, or how data is to be transformed (functions). Components are the central building pieces of

any mapping. On the mapping area, components appear as rectangles. The following are examples of MapForce components:

- Constants
- Filters
- Conditions
- Function components
- EDI documents (UN/EDIFACT, ANSI X12, HL7)
- Excel 2007+ files
- Simple input components
- Simple output components
- XML Schemas and DTDs

Connector

A connector is a small triangle displayed on the left or right side of a component. The connectors displayed on the left of a component provide data entry points *to that component*. The connectors displayed on the right of a component provide data exit points *from that component*.

Connection

A connection is a line that you can draw between two connectors. By drawing connections, you instruct MapForce to transform data in a specific way (for example, read data from an XML document and write it to another XML document).

Source component

A source component is a component from which MapForce reads data. When you run the mapping, MapForce reads the data supplied by the connector of the source component, converts it to the required type, and sends it to the connector of the target component.

Target component

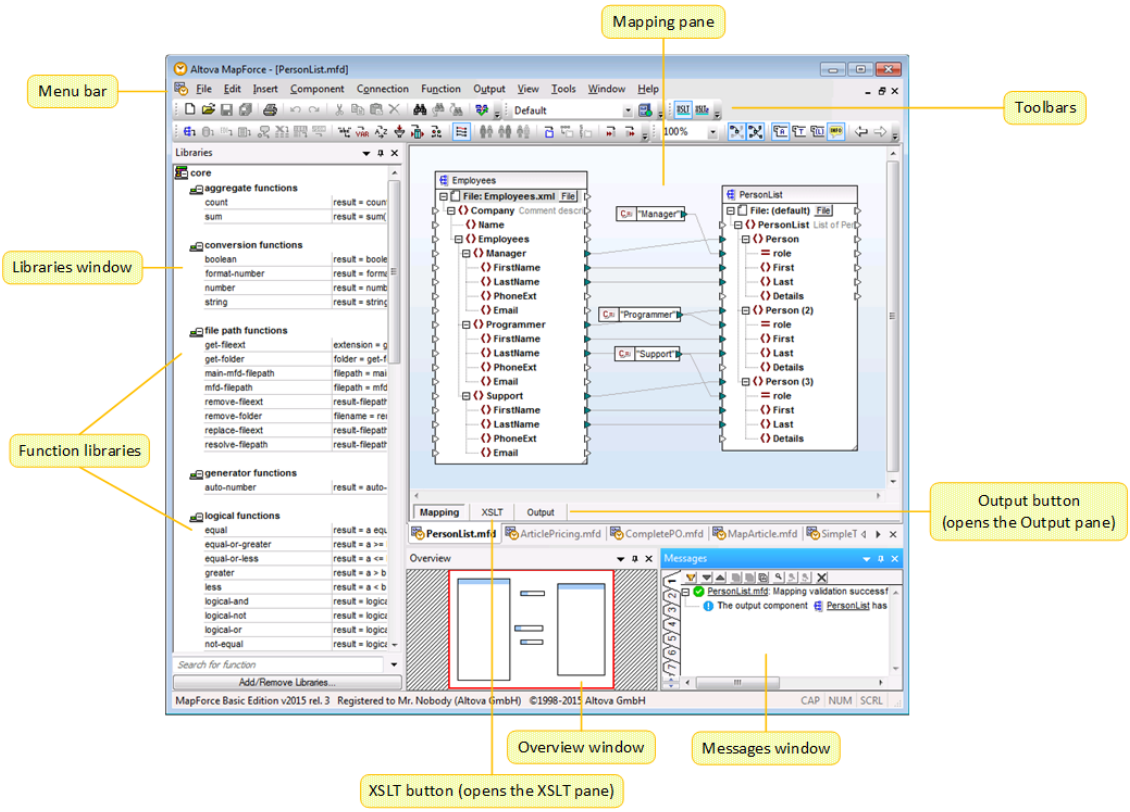
A target component is a component to which MapForce writes data. When you run the mapping, a target component instructs MapForce to either generate a file (or multiple files) or output the result as a string value for further processing in an external program. A target component is the opposite of a source component.

© 2015 Altova GmbH

2.3 User Interface Overview

The graphical user interface of MapForce is organized as an integrated development environment. The main interface components are illustrated below. You can change the interface settings by using the menu command **Tools | Customize**.

Use the ▼ 📌 ✕ buttons displayed in the upper-right corner of each window to show, hide, pin, or dock it. If you need to restore toolbars and windows to their default state, use the menu command **Tools | Restore Toolbars and Windows**.



MapForce graphical user interface (MapForce Basic Edition)

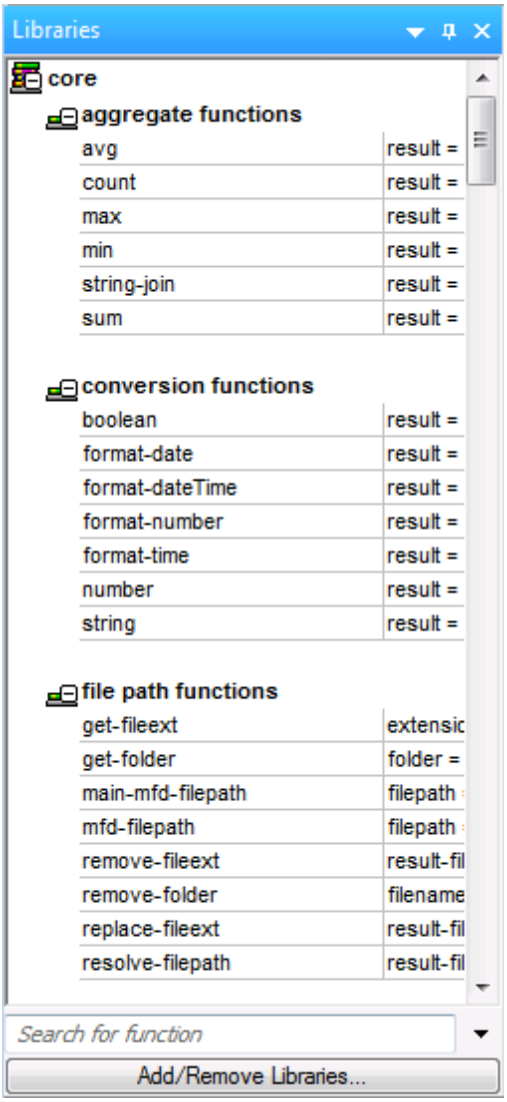
Menu Bar and Toolbars

The Menu Bar displays the menu items. Each toolbar displays a group of buttons representing MapForce commands. You can reposition the toolbars by dragging their handles to the desired locations.

Libraries window

The Libraries window lists the MapForce built-in functions, organized by library. The list of available functions changes based on the transformation language you select. If you have

created user-defined functions, or if you imported external libraries, they also appear in the Libraries window.



Mapping pane

The Mapping pane is the working area where you design mappings. You can add mapping components (such as files, schemas, constants, variables, and so on) to the mapping area from the **Insert** menu. You can also drag into the Mapping pane functions displayed in the Libraries window.

XSLT pane

The XSLT pane displays the XSLT 1.0 transformation code generated from your mapping, when you click the **XSLT** button. This pane is available when you select XSLT as transformation language.

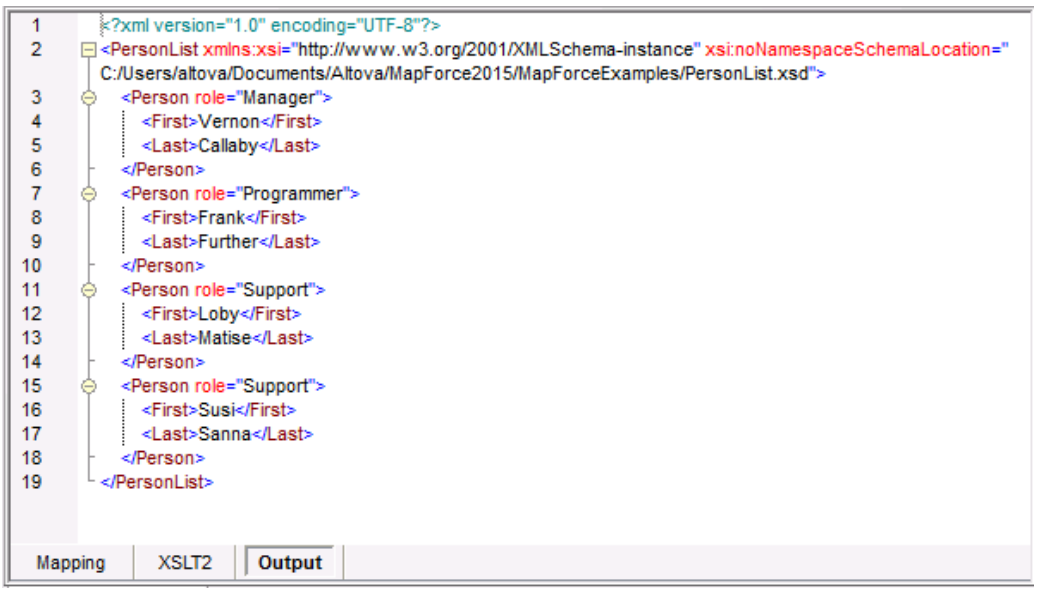
 © 2015 Altova GmbH

XSLT2 pane

The XSLT2 pane displays the XSLT 2.0 transformation code generated from your mapping, when you click the **XSLT2** button. This pane is available when you select XSLT2 as transformation language.

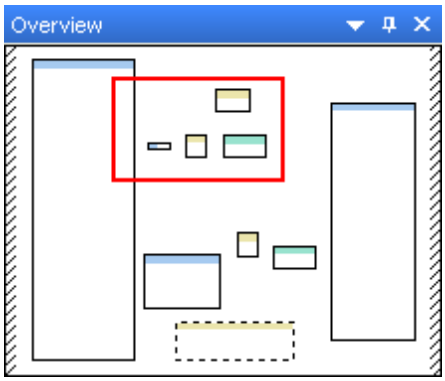
Output pane

The Output pane displays the result of the mapping transformation (for example, an XML file), when you click the **Output** button. If the mapping generates multiple files, you can navigate sequentially through each generated file.



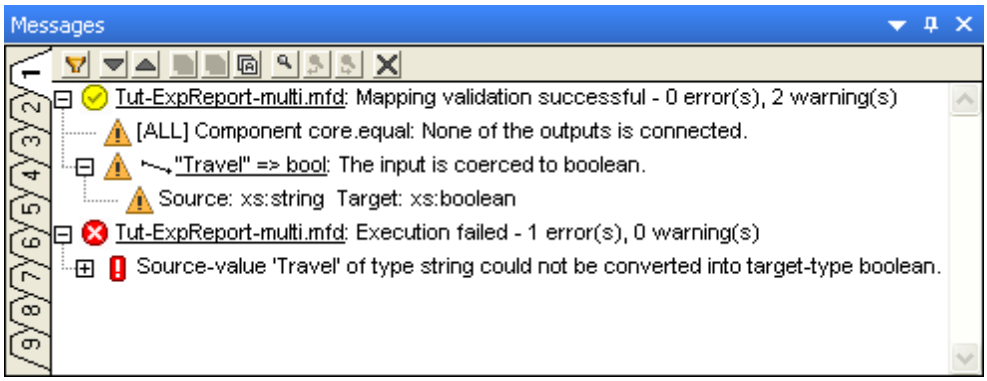
Overview window

The Overview window gives a bird's-eye view of the Mapping pane. Use it to navigate quickly to a particular location on the mapping area when the size of the mapping is very large. To navigate to a particular location on the mapping, click and drag the red rectangle.



Messages window

The Messages window shows messages, errors, and warnings when you preview a mapping (by clicking the **Output** button) or perform a mapping validation. You can copy the output messages to the clipboard if required.


Messages
Tut-ExpReport-multi.mfd: Mapping validation successful - 0 error(s), 2 warning(s)
[ALL] Component core.equal: None of the outputs is connected.
"Travel" => bool: The input is coerced to boolean.
Source: xs:string Target: xs:boolean
Tut-ExpReport-multi.mfd: Execution failed - 1 error(s), 0 warning(s)
Source-value 'Travel' of type string could not be converted into target-type boolean.

Application status bar

The application status bar appears at the bottom of the application window, and shows application-level information. The most useful of this information are the tooltips that are displayed here when you move the mouse over a toolbar button. If you are using the 64-bit version of MapForce, the application name appears in the status bar with the suffix (x64). There is no suffix for the 32-bit version.

© 2015 Altova GmbH

2.4 Conventions

Example files

Most of the data mapping design files (files with .mfd extension, as well as other accompanying instance files) illustrated or referenced in this documentation are available in the following folders:

- (My) Documents\Altova\MapForce2016\MapForce Examples
- (My) Documents\Altova\MapForce2016\MapForce Examples\Tutorials

The location of **(My) Documents** folder depends on your version of Windows.

Windows 10 Windows 8 Windows 7 Windows Vista Windows Server 2008 Windows Server 2012	`C:\Users\<username>\Documents`
Windows XP Windows Server 2003	`C:\Documents and Settings\<username>\My Documents`

The example mappings and instance files accompanying MapForce illustrate most aspects of how it works, and you are highly encouraged to experiment with them as you learn about MapForce. When in doubt about the possible effects of making changes to the MapForce original examples, create back-ups before changing them.

Graphical user interface

Some of the images (screen shots) accompanying into this documentation depict graphical user interface elements that may not be applicable to your MapForce edition. In relevant contexts, images will typically include the name of the source mapping design (*.mfd) file, as well as the edition of MapForce in which the graphic was produced.

Chapter 3

Tutorials

3 Tutorials

The MapForce tutorials are intended to help you understand and use the basic data transformation capabilities of MapForce in a short amount of time. You can regard these tutorials as a "crash course" of MapForce. While the goal is not to illustrate completely all MapForce features, you will be guided through the MapForce basics step-by-step, so it is recommended that you follow the tutorials sequentially. It is important that you understand each concept before moving on to the next one, as the tutorials gradually grow in complexity. Basic knowledge of XML and XML schema will be advantageous.

Convert XML to New Schema

This tutorial shows you how to convert data from an XML structure to another using the XSLT 2.0 language, without writing any code. You will also learn about MapForce sequences and items, creating mapping connections, using a function, validating and previewing a mapping, as well as saving the resulting output to the disk.

Map Multiple Sources to One Target

This tutorial shows you how to read data from two XML files with different schema and merge it into a single target XML file. You will also learn how to change the name and instance files of each mapping component, and the concept of "duplicate inputs".

Work with Multiple Target Schemas

This tutorial shows you how to work with more complex mapping that produce two or more target outputs. More specifically, you will learn how to generate, in the same mapping, an XML file that stores a list of book records, and another XML file that contains only a subset of the books in the first file, filtered by a specific publication year. To support filtering data, you will use a **Filter** component, a function and a numeric constant.

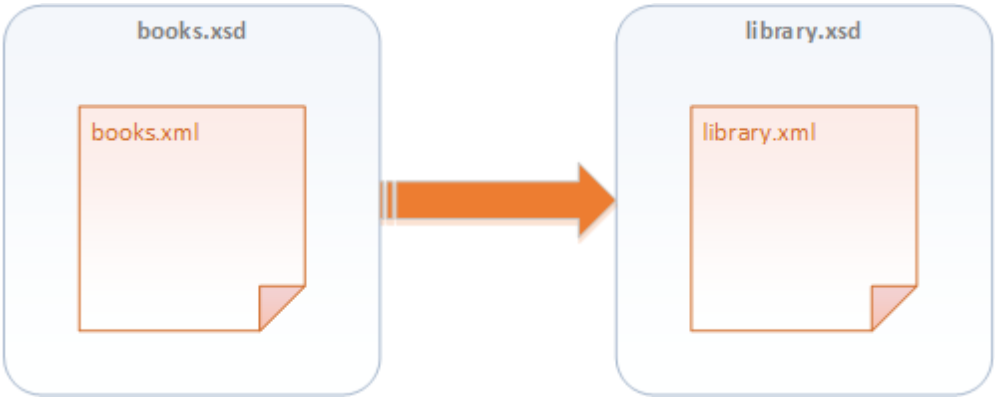
Process and Generate Files Dynamically

This tutorial shows you how to read data from multiple XML instance files located in the same folder and write it to multiple XML files generated on the fly. You will also learn about stripping the XML and schema declarations and using functions to concatenate strings and extract file extensions.

© 2015 Altova GmbH

3.1 Convert XML to New Schema

This tutorial shows you how to convert data between two XML files, while helping you learn the basics of the MapForce development environment. Both XML files store a list of books, but their elements are named and organized in a slightly different way (that is, the two files have different schemas).



Abstract model of the data transformation

The code listing below shows sample data from the file that will be used as data source (for the sake of simplicity, the XML and the namespace declarations are omitted).

```
<books>
   <book id="1">
      <author>Mark Twain</author>
      <title>The Adventures of Tom Sawyer</title>
      <category>Fiction</category>
      <year>1876</year>
   </book>
   <book id="2">
      <author>Franz Kafka</author>
      <title>The Metamorphosis</title>
      <category>Fiction</category>
      <year>1912</year>
   </book>
</books>
```

books.xml

This is how data should look in the target (destination) file:

```
<library>
   <last_updated>2015-06-02T16:26:55+02:00</last_updated>
   <publication>
      <id>1</id>
      <author>Mark Twain</author>
```

```
      <title>The Adventures of Tom Sawyer</title>
      <genre>Fiction</genre>
      <publish_year>1876</publish_year>
   </publication>
   <publication>
      <id>2</id>
      <author>Franz Kafka</author>
      <title>The Metamorphosis</title>
      <genre>Fiction</genre>
      <publish_year>1912</publish_year>
   </publication>
</library>
```

library.xml

As you may have noticed, some element names in the source and target XML are not the same. Our goal is to populate the `<author>`, `<title>`, `<genre>` and `<publish_year>` elements of the target file from the equivalent elements in the source file (`<author>`, `<title>`, `<category>`, `<year>`). The attribute `id` in the source XML file must be mapped to the `<id>` element in the target XML file. Finally, we must populate the `<last_updated>` element of the target XML file with the date and time when the file was last updated.

To achieve the required data transformation, let's take the following steps.

Step 1: Select XSLT2 as transformation language

You can do this in one of the following ways:

- Click the **XSLT2** ( ) toolbar button.
- On the **Output** menu, click **XSLT 2.0**.

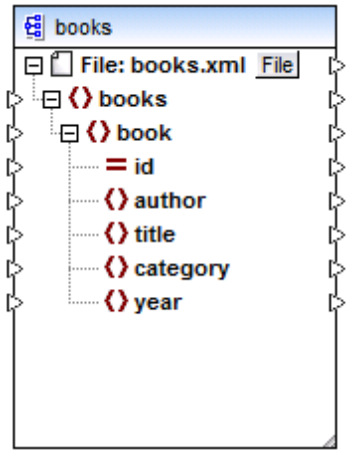
Step 2: Add the source XML file to the mapping

The source XML file for this mapping is located at the following path: **<Documents>\Altova \MapForce2016\MapForceExamples\Tutorial\books.xml**. You can add it to the mapping in one of the following ways:

- Click the **Insert XML Schema/File** ( ) toolbar button.
- On the **Insert** menu, click **XML Schema/File**.
- Drag the XML file from Windows Explorer into the mapping area.

Now that the file has been added to the mapping area, you can see its structure at a glance. In MapForce, this structure is known as a mapping component, or simply component. You can expand elements in the component either by clicking the collapse (⊟) and expand icons (⊞), or by pressing the **+** and **-** keys on the numeric keypad.

© 2015 Altova GmbH



Mapping component

To move the component inside the mapping pane, click the component header and drag the mouse to a new position. To resize the component, drag the corner of the component . You can also double-click the corner so that MapForce adjusts the size automatically.

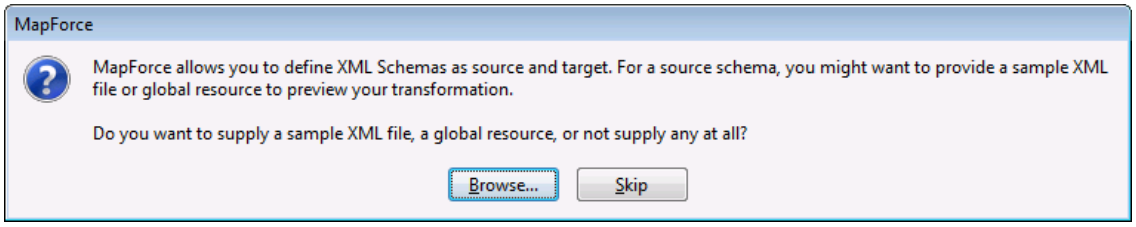
The top level node represents the file name; in this particular case, its title displays the name of the XML instance file. The XML elements in the structure are represented by the icon, while XML attributes are represented by the icon.

The small triangles displayed on both sides of the component represent data inputs (if they are on the left side) or outputs (when they are on the right side). In MapForce, they are called input connectors and output connectors, respectively.

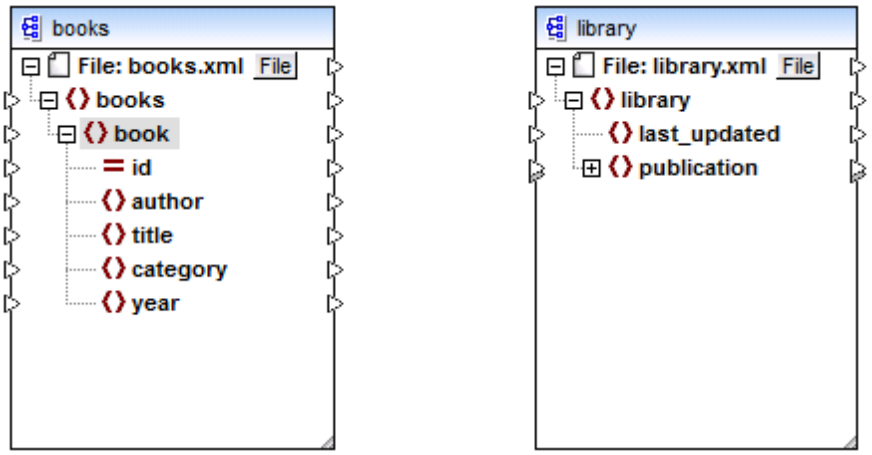
Step 3: Add the target XML schema to the mapping

To generate the target XML, we will use an existing XML schema file. In a real-life scenario, this file may have been provided to you by a third party, or you can create it yourself with a tool such as XMLSpy. If you don't have a schema file for your XML data, MapForce prompts you to generate it whenever you add to the mapping an XML file without an accompanying schema or schema reference.

For this particular example, we are using an existing schema file available at: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\library.xsd**. To add it to the mapping, follow the same steps as with the source XML file (that is, click the **Insert XML Schema/File** ( ) toolbar button). Click **Skip** when prompted by MapForce to supply an instance file.


MapForce
MapForce allows you to define XML Schemas as source and target. For a source schema, you might want to provide a sample XML file or global resource to preview your transformation.
Do you want to supply a sample XML file, a global resource, or not supply any at all?
Browse...
Skip

At this stage, the mapping design looks as follows:

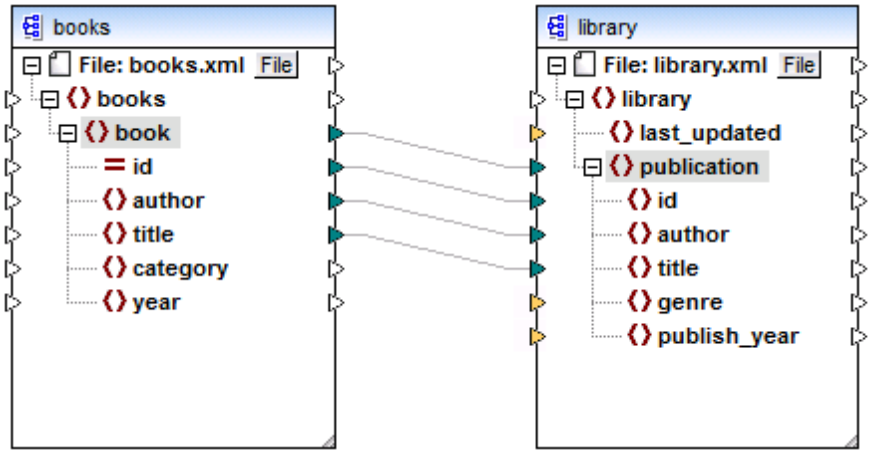


Step 4: Make the connections

For each <book> in the source XML file, we want to create a new <publication> in the target XML file. We will therefore create a mapping connection between the <book> element in the source component and the <publication> element in the target component. To create the mapping connection, click the output connector (the small triangle) to the right of the <book> element and drag it to the input connector of the <publication> element in the target.

When you do this, MapForce may automatically connect all elements which are children of <book> in the source file to elements having the same name in the target file; therefore, four connections are being created simultaneously. This behavior is called "Auto Connect Matching Children" and it can be disabled and customized if necessary.

 © 2015 Altova GmbH



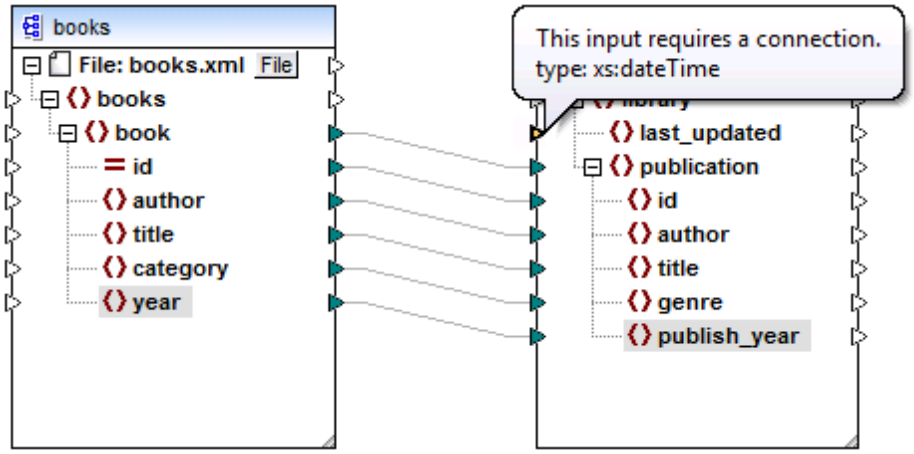
You can enable or disable the "Auto Connect Matching Children" behavior in one of the following ways:

- Click the **Toggle auto connect of children** ( ) toolbar button.
- On the **Connection** menu, click **Auto Connect Matching Children**.

Notice that some of the input connectors on the target component have been highlighted by MapForce in orange, which indicates that these items are mandatory. To ensure the validity of the target XML file, provide values for the mandatory items as follows:

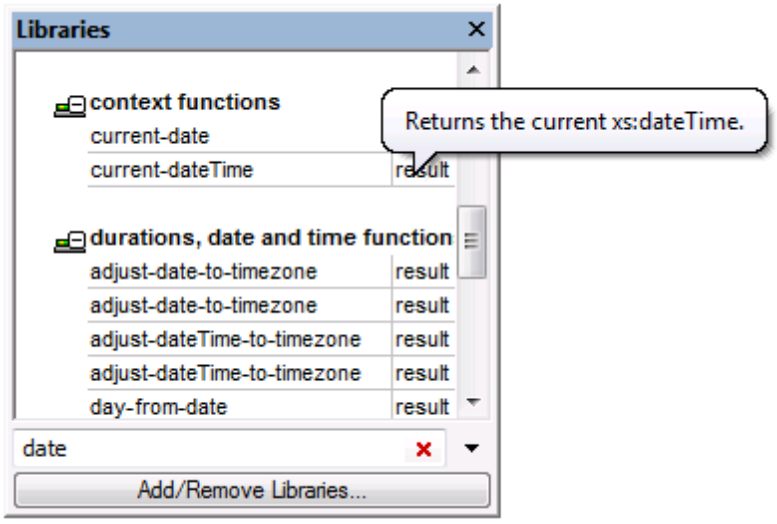
- Connect the `<category>` element in the source with the `<genre>` element in the target
- Connect the `<year>` element in the source with the `<publish_year>` element in the target

Finally, you need to supply a value to the `<last_updated>` element. If you move the mouse over its input connector, you can see that the element is of type `xs:dateTime`. Note that, for tips to be displayed, the **Show tips** ( ) toolbar button must be enabled.



You can also make the data type of each item visible at all times, by clicking the **Show Data Types** ( ) toolbar button.

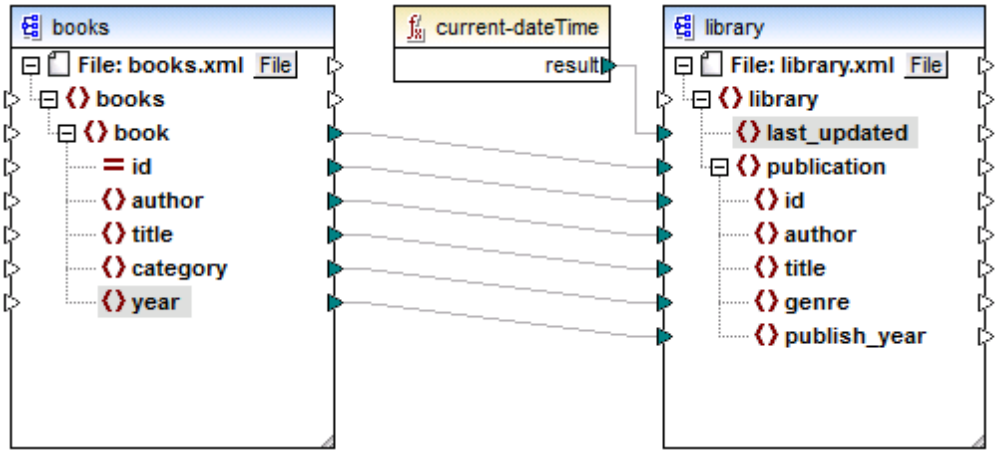
You can get the current date and time (that is, the `xs:dateTime` value) by means of a date and time XSLT function. To find the XSLT function to the mapping, start typing "date" in the text box located in the lower part of the Libraries window.



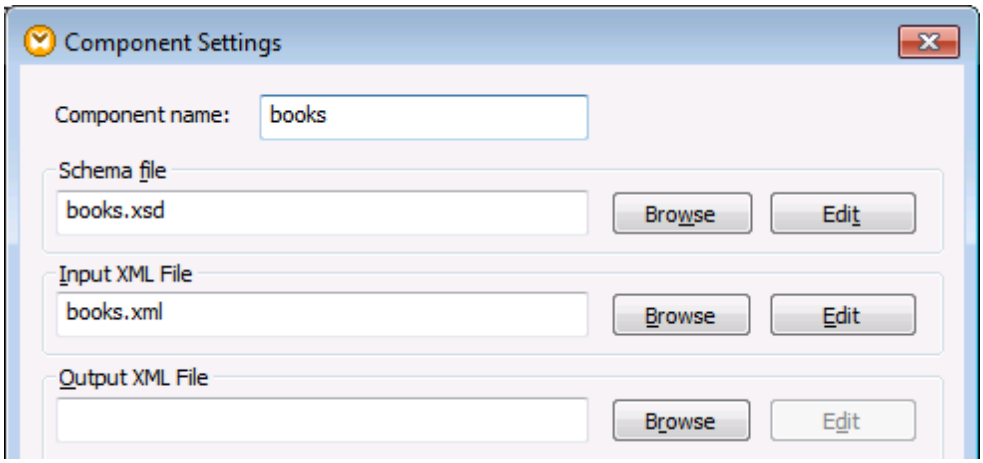
As shown above, if you move the mouse over the "result" part of the function, you can see its description. For tips to be displayed, make sure that the **Show tips** ( ) toolbar button is enabled.

To add the function to the mapping, drag the function into the mapping pane, and connect its output to the input of the `<last_updated>` element.

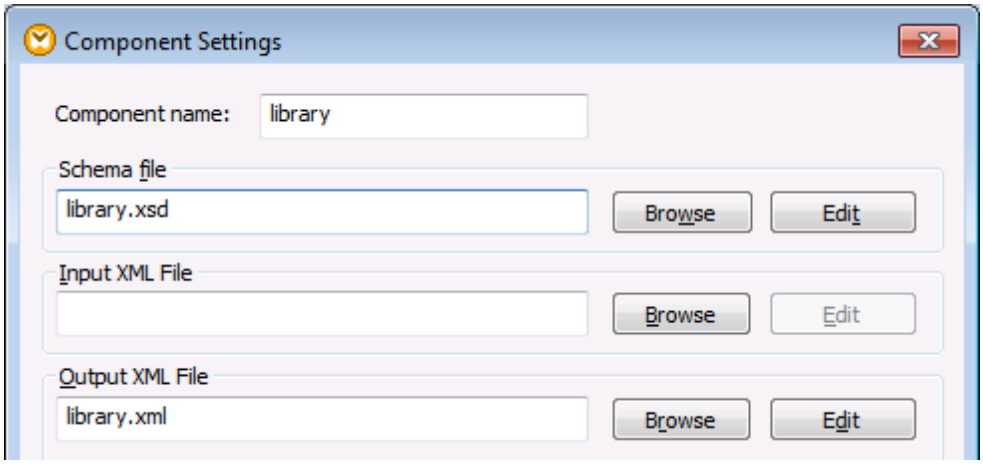
 © 2015 Altova GmbH



You have now created a MapForce mapping design (or simply a "mapping") which converts data from the **books.xml** instance file (having the **books.xsd** schema) to the new **library.xml** file (having the **library.xsd** schema). If you double-click the header of each component, you can view these and other settings in the Component Settings dialog box, as shown below.



Component settings for the source



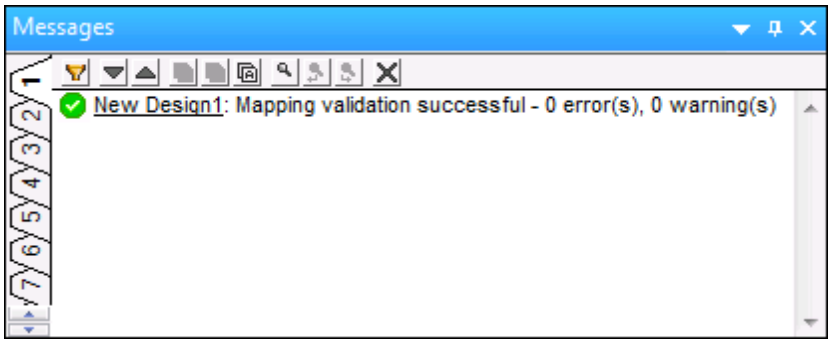
Component settings for the target

Step 5: Validate and save the mapping

Validating a mapping is an optional step that enables you to see and correct potential mapping errors and warnings before you run the mapping. To check whether the mapping is valid, do one of the following:

- On the **File** menu, click **Validate Mapping**.
- Click the **Validate** ( ) toolbar button.

The Messages window displays the validation results:



Messages window

At this point, you might also want to save the mapping to a file. To save the mapping, do one of the following:

- On the **File** menu, click **Save**.
- Click the **Save** ( ) toolbar button.

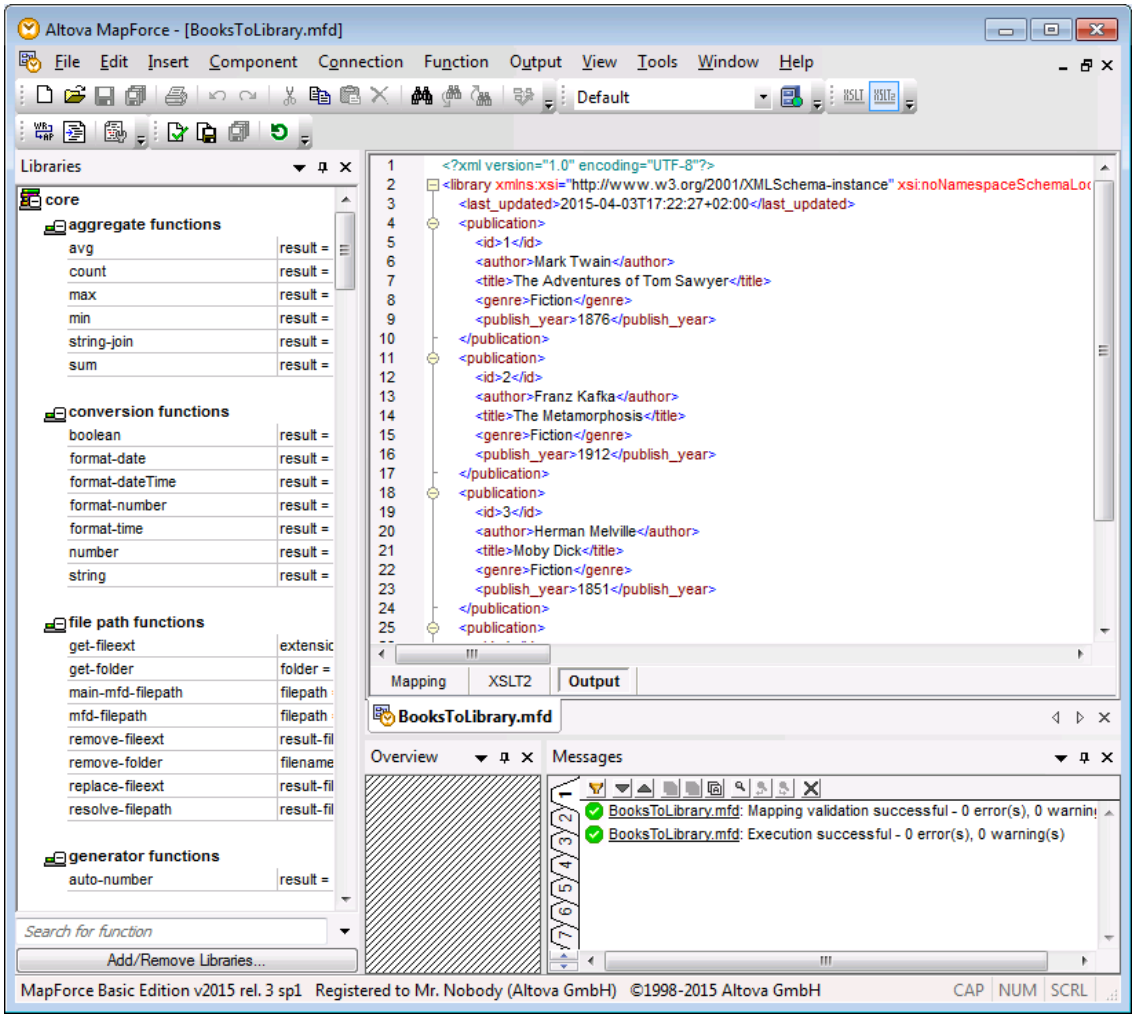
For your convenience, the mapping created in this tutorial is available at the following path:

 © 2015 Altova GmbH

<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\\BooksToLibrary.mfd. Therefore, from this point onwards, you can either continue with the mapping file you created, or with the **BooksToLibrary.mfd** file.

Step 6: Preview the mapping result

You can preview the result of the mapping directly in MapForce. To do this, click the **Output** button located in the lower part of the mapping pane. MapForce runs the transformation and displays the result of the mapping in the **Output** pane.



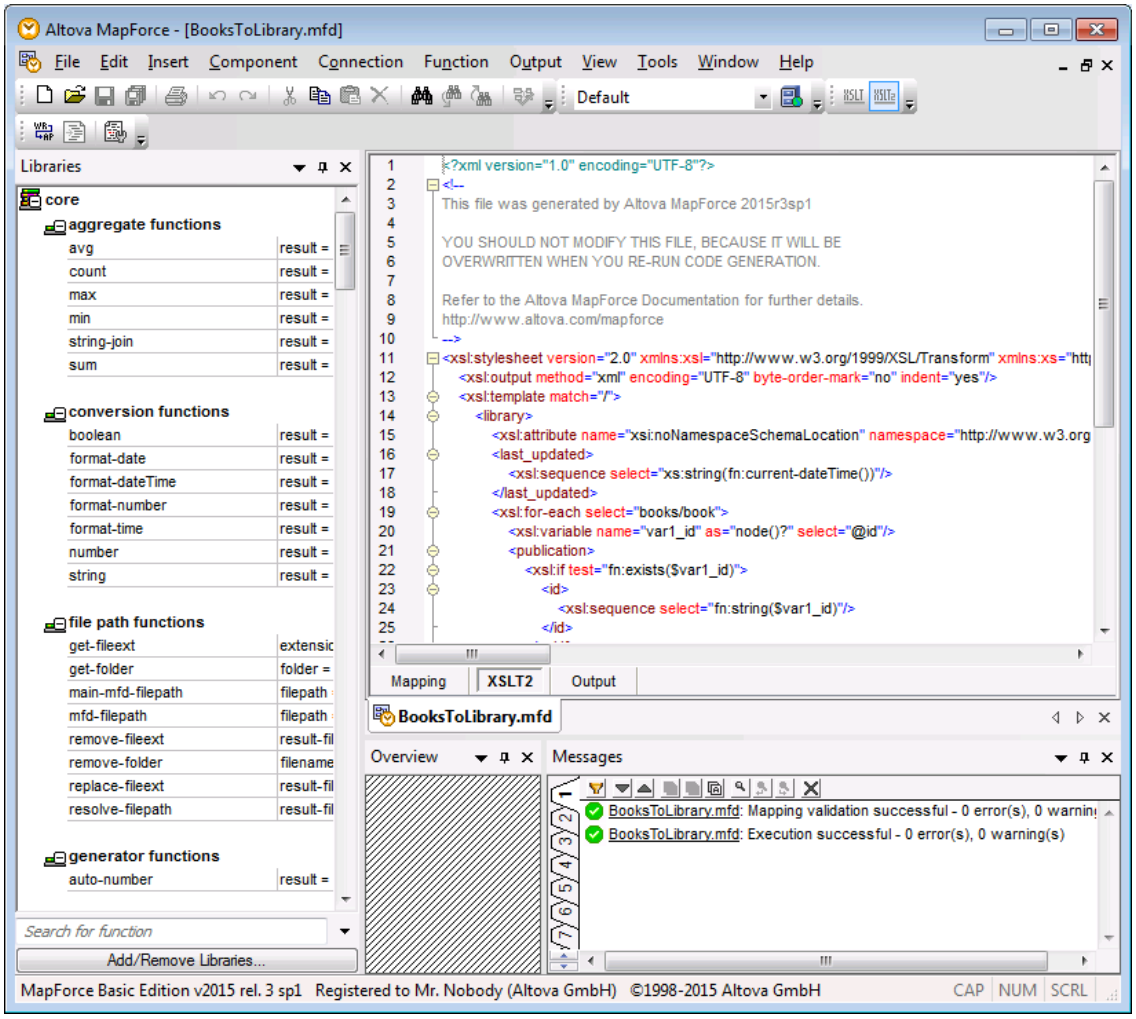
Output pane

You can now see the result of the transformation in MapForce.

By default, the files displayed for preview in the **Output** pane are not written to the disk. Instead, MapForce creates temporary files. To save the file displayed in the **Output** pane to the disk, select the menu command **Output | Save Output File**, or click the **Save generated output** ( ) toolbar button.

To configure MapForce to write the output directly to final files instead of temporary, go to **Tools | Options | General**, and then select the **Write directly to final output files** check box. Note that enabling this option is not recommended while you follow this tutorial, because you may unintentionally overwrite the original tutorial files.

You can also preview the generated XSLT code that performs the transformation. To preview the code, click the **XSLT2** button located in the lower area of the mapping pane.



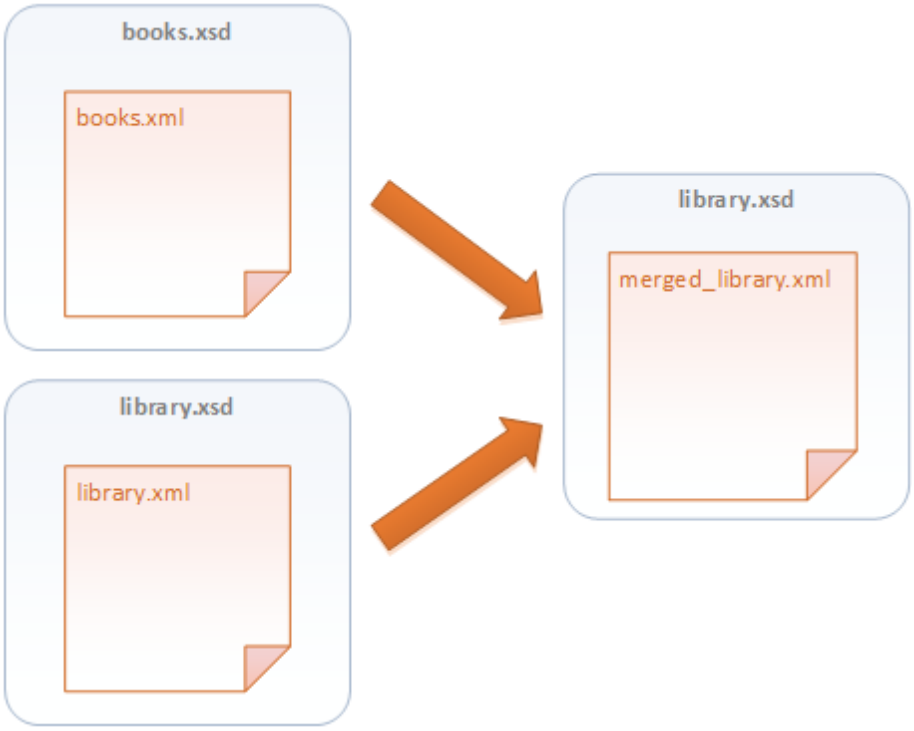
XSLT2 pane

To generate and save the XSLT2 code to a file, select the menu item **File | Generate Code in | XSLT 2.0**. When prompted, select a folder where the generated code must be saved. After code generation completes, the destination folder includes the following two files:

1. An XSLT transformation file, named after the target schema (in this example, **MappingMaptolibrary.xslt**).
2. A **DoTransform.bat** file. The **DoTransform.bat** file enables you to run the XSLT transformation in RaptorXML Server (for more information, see http://www.altova.com/raptorxml.html).

3.2 Map Multiple Sources to One Target

In the previous tutorial, you have converted data from a source file (**books.xml**) to a target file (**library.xml**). The target file (**library.xml**) did not exist before running the mapping; it was generated by the mapping transformation. Let's now imagine a scenario where you already have some data in the **library.xml** file, and you want to merge this data with data converted from the **books.xml**. The goal in this tutorial is to design a mapping that generates a file called **merged_library.xml**. The generated file will include data from two sources: the **books.xml** file and the **library.xml** file. Note that the files used as source (**books.xml** and **library.xml**) have different schemas. If the source files had the same schema, you could also merge their data using a different approach (see Process and Generate Files Dynamically).



Abstract model of the data transformation

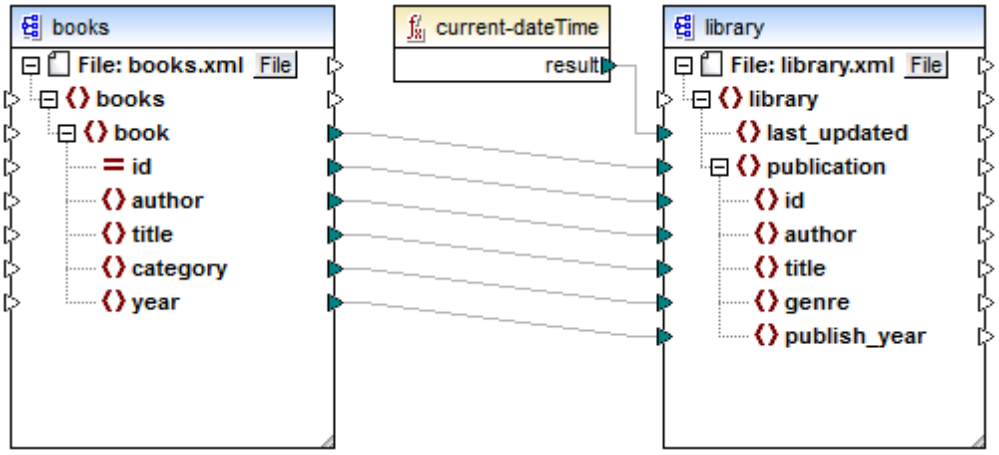
To achieve the required goal, let's take the following steps.

Step 1: Prepare the mapping design file

This tutorial uses as starting point the **BooksToLibrary.mfd** mapping from the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder. You have already designed this mapping in the Convert XML to New Schema tutorial. To begin, open the **BooksToLibrary.mfd** file in MapForce, and save it with a new name.

> Make sure to save the new mapping in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder, because it references several files from it.

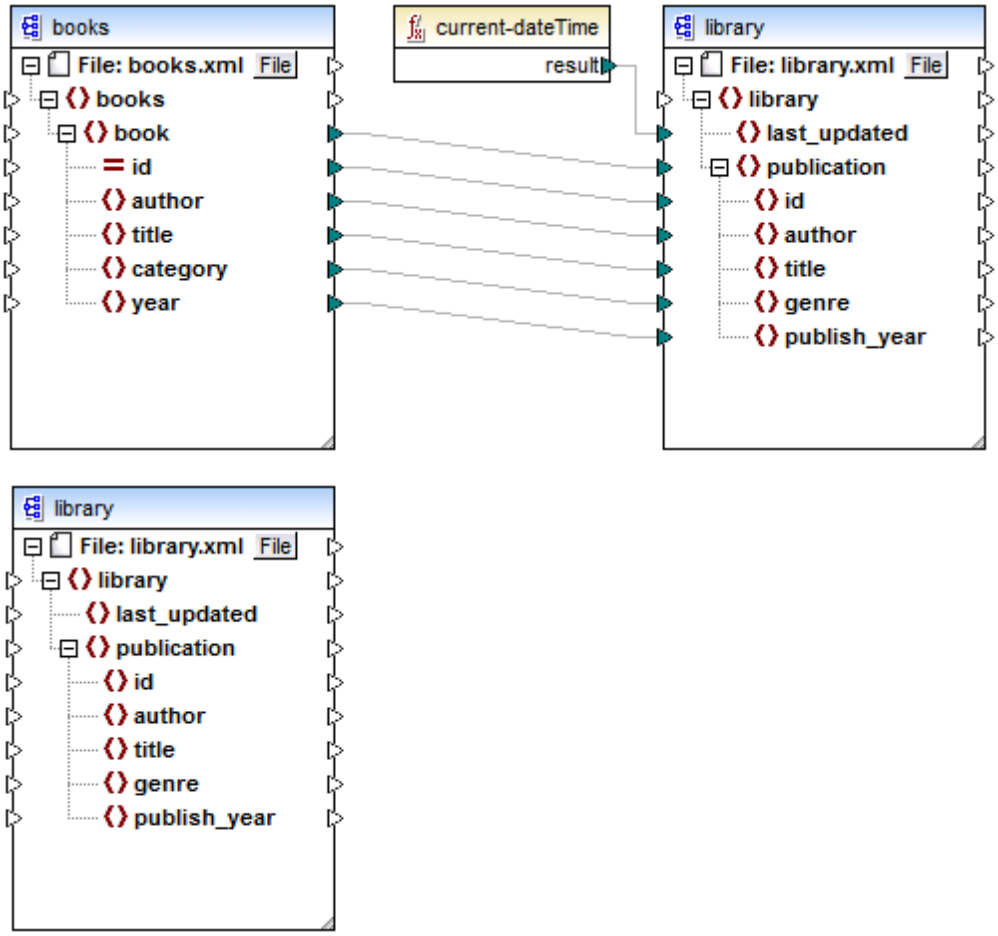
 © 2015 Altova GmbH



BooksToLibrary.mfd (MapForce Basic Edition)

Step 2: Create a second source component

First, select the target component and copy it (press **Ctrl + C**), and then paste it (press **Ctrl + V**) into the same mapping. Click the header of the new component and drag it under the **books** component.



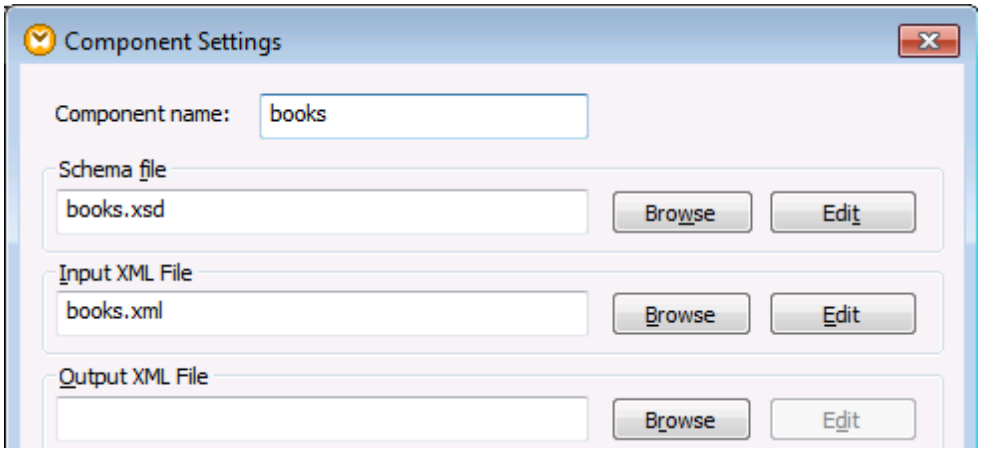
The mapping now has two source components: **books** and **library**, and one target component: **library**.

> You can always move the mapping components in any direction (left, right, top, bottom). Nevertheless, placing a source component to the left of a target component will make your mapping easier to read and understand by others. This is also the convention for all mappings illustrated in this documentation, as well as in the sample mapping files accompanying your MapForce installation.

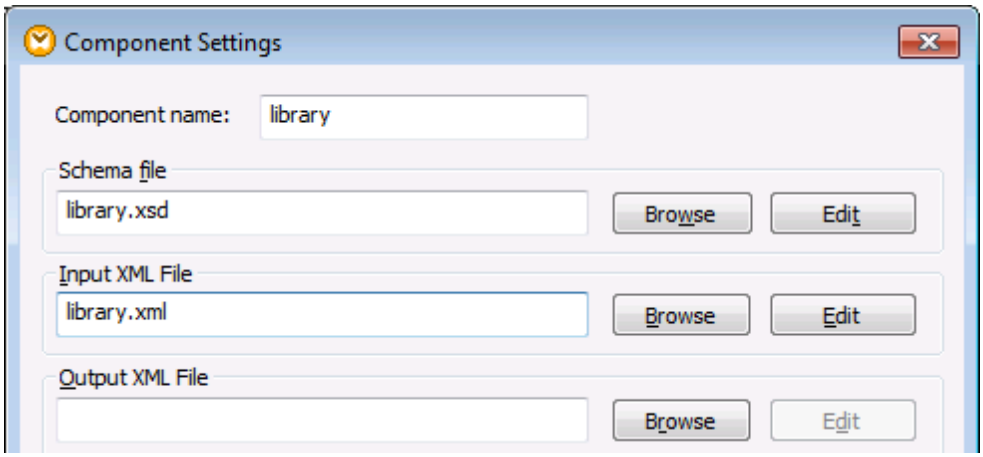
Step 3: Verify and set the input/output files

In the previous step, the new source component was copy-pasted from the target component, so it inherits the same settings. To ensure that the name input/output instance files are correctly set, double-click the header of each component, and, in the Component Settings dialog box, verify and change the name and the input/output files of each component as shown below.

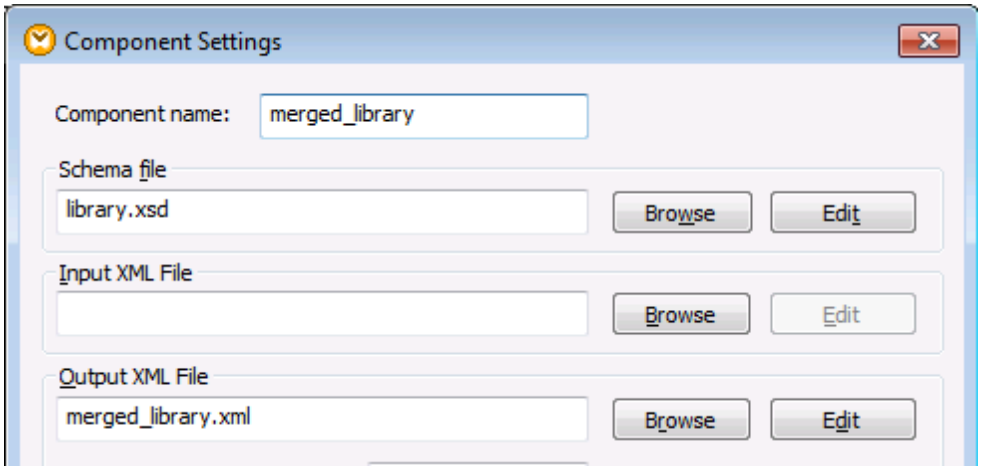
 © 2015 Altova GmbH



*Components settings for the first source (**books**)*



*Component settings for the second source (**library**)*



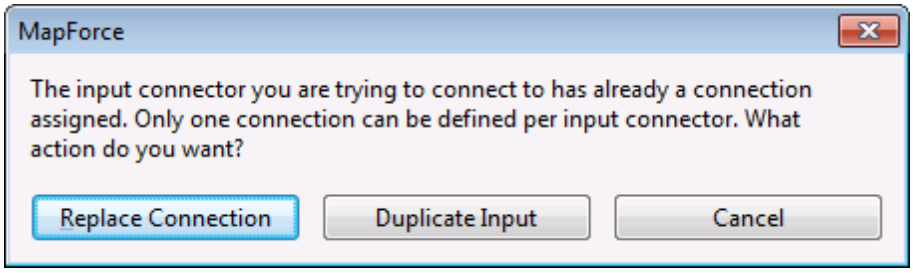
*Component settings for the target (**merged_library**)*

As shown above, the first source component reads data from **books.xml**. The second source component reads data from **library.xml**. Finally, the target component outputs data to a file called **merged_library.xml**.

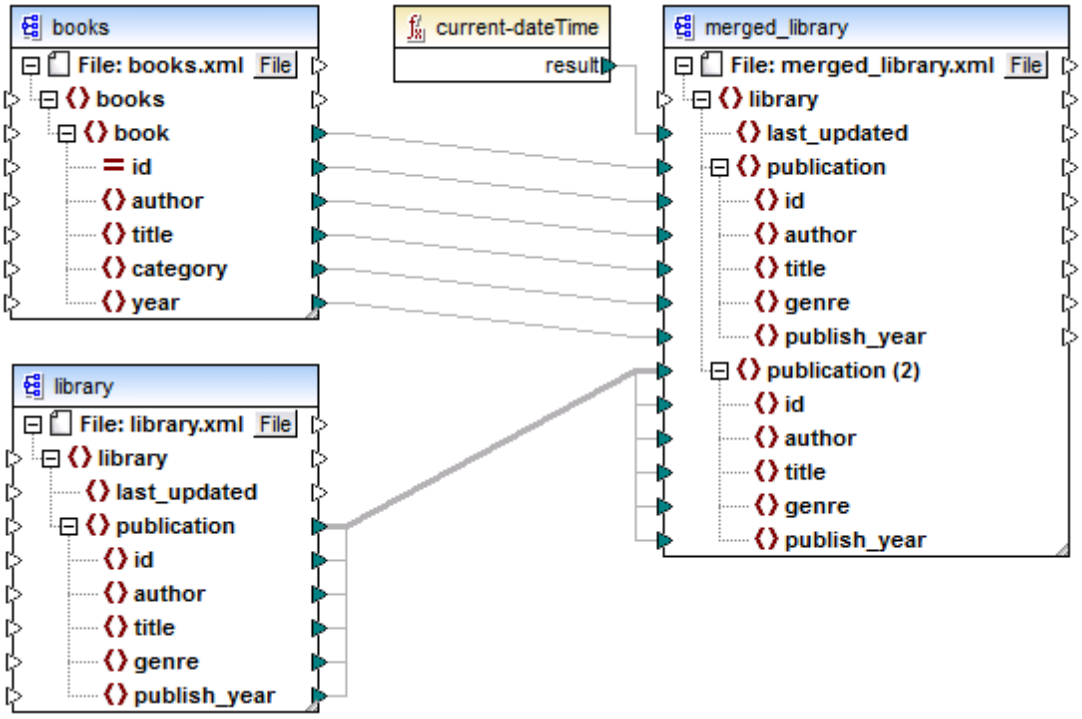
Step 4: Make the connections

To instruct MapForce to write data from the second source to the target, click the output connector (small triangle) of the `publications` item in the source **library** component and drag it to the input connector of the publications item in the target **library** component. Because the target input connector already has a connection to it, the following notification message appears.

© 2015 Altova GmbH


MapForce
The input connector you are trying to connect to has already a connection assigned. Only one connection can be defined per input connector. What action do you want?
Replace Connection
Duplicate Input
Cancel

In this particular tutorial, replacing the connection is not what we want to achieve; our goal is to map data from two sources. Therefore, click **Duplicate Input**. By doing so, you configure the target component to accept data from the new source as well. The mapping now looks as follows:



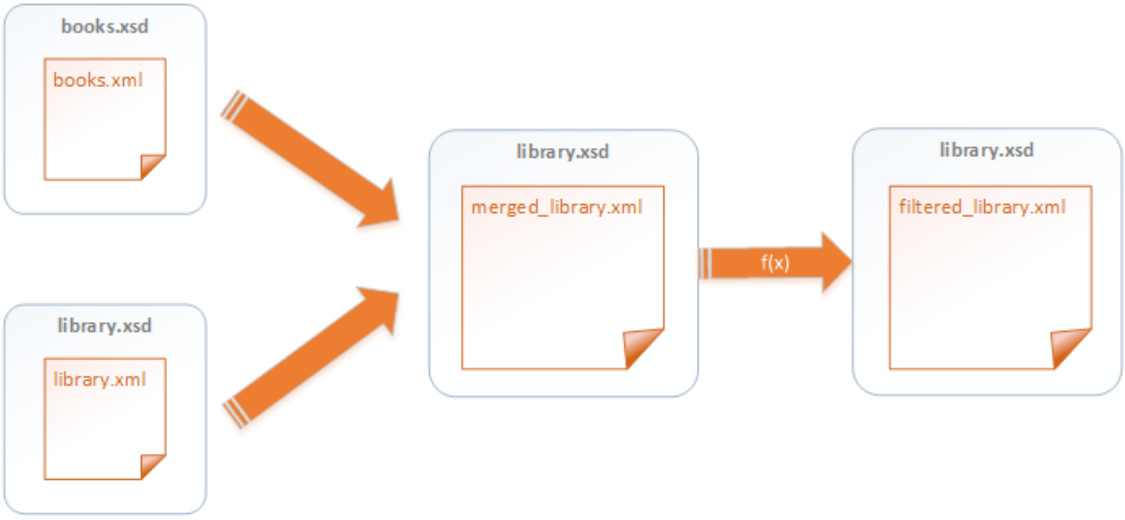
Notice that the `publication` item in the target component has now been duplicated. The new `publication(2)` node will accept data from the source **library** component. Importantly, even though the name of this node appears as `publication(2)` in the mapping, its name in the resulting XML file will be `publication`, which is the intended goal.

You can now click the **Output** button at the bottom of the mapping pane, and view the mapping result. You will notice that data from both **library.xml** and **books.xml** files has now been merged into the new **merged_library.xml** file.

3.3 Work with Multiple Target Schemas

In the previous tutorial, Map Multiple Sources to One Target, you have seen how to map data from multiple source schemas to a single target schema. You have also created a file called **merged_library.xml**, which stores book records from two sources. Now let's assume that someone from another department has asked you to provide a subset of this XML file. Specifically, you must deliver an XML file that includes only the books published after 1900.

For convenience, you can modify the existing **MultipleSourcesToOneTarget.mfd** mapping so that, whenever required, you can generate both the complete XML library, and the filtered library.



Abstract model of the data transformation

In the diagram above, the data is first merged from two different schemas (**books.xsd** and **library.xsd**) into a single XML file called **merged_library.xml**. Secondly, the data is transformed using a filtering function and passed further to the next component, which creates an XML file called **filtered_library.xml**. The "intermediate" component acts both as data target and source. In MapForce, this technique is known as "chaining mappings", which is also the subject of this tutorial.

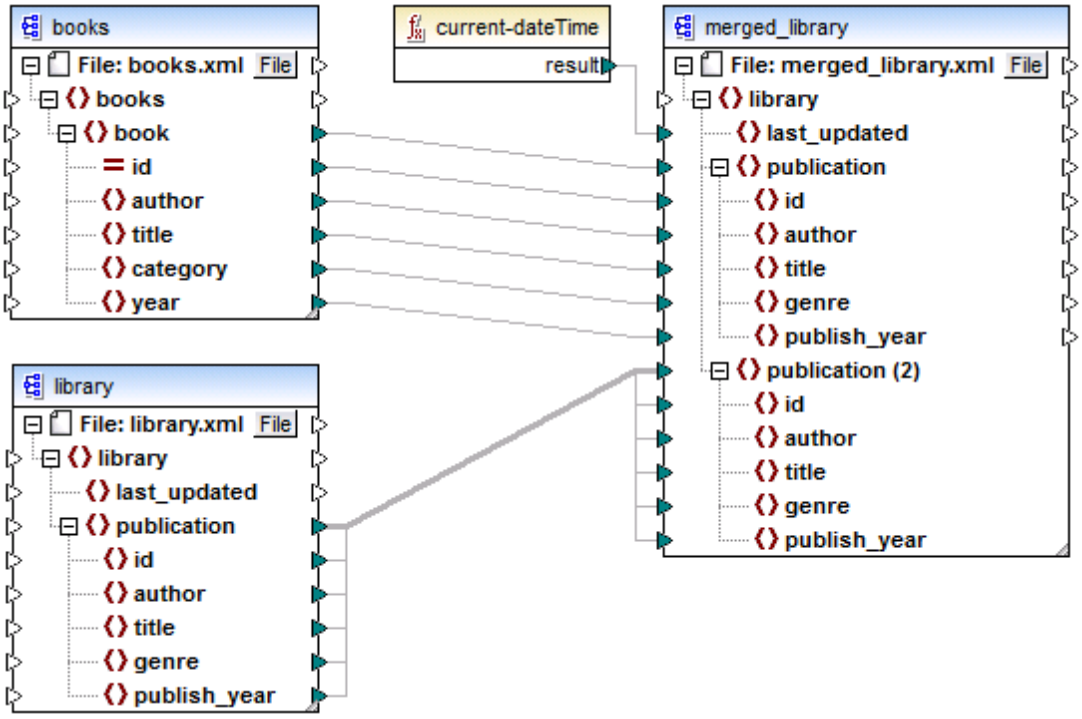
Our goal is to make it possible to generate at any time both the **merged_library.xml** and the **filtered_library.xml**. To achieve the goal, let's take the following steps.

Step 1: Prepare the mapping design file

This tutorial uses as starting point the **MultipleSourcesToOneTarget.mfd** mapping from the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder. You have already designed this mapping in the Map Multiple Sources to One Target tutorial. To begin, open the **MultipleSourcesToOneTarget.mfd** file in MapForce, and save it with a new name.

 © 2015 Altova GmbH

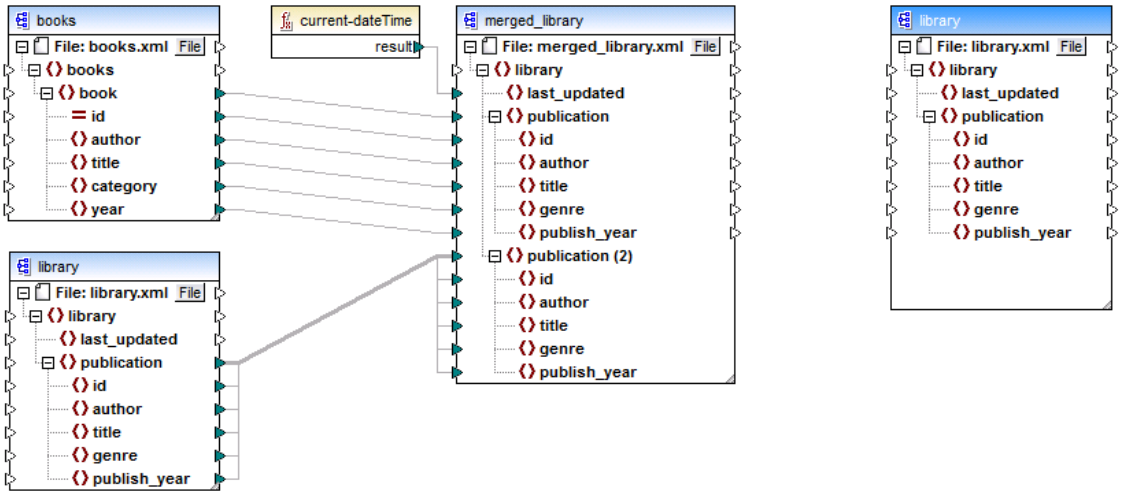
Make sure to save the new mapping in the **<Documents>\Altova\MapForce2016 \MapForceExamples\Tutorial** folder, because it references several files from it.



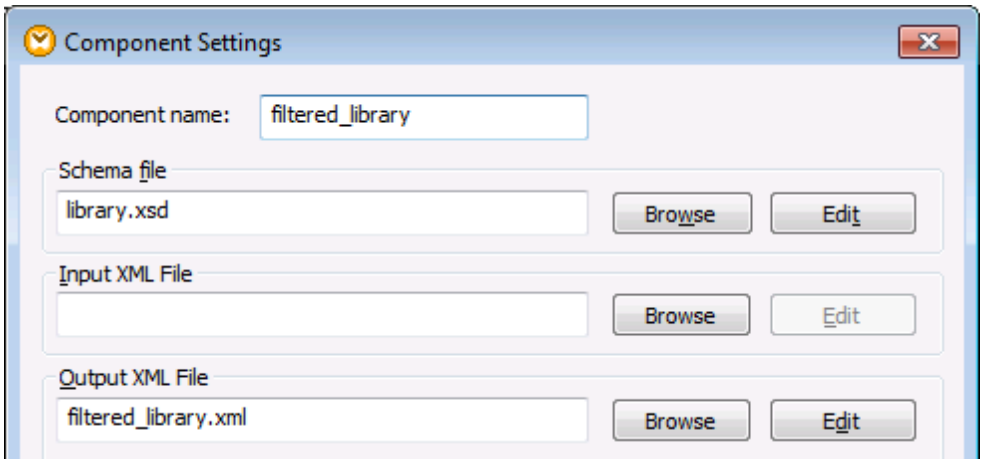
MultipleSourcesToOneTarget.mfd (MapForce Basic Edition)

Step 2: Add and configure the second target component

To add the second target component, click the **Insert XML Schema/File** ( ) toolbar button, and open the **library.xsd** file located in the **<Documents>\Altova\MapForce2016 \MapForceExamples\Tutorial** folder. Click **Skip** when prompted to supply a sample instance file. The mapping now looks as follows:

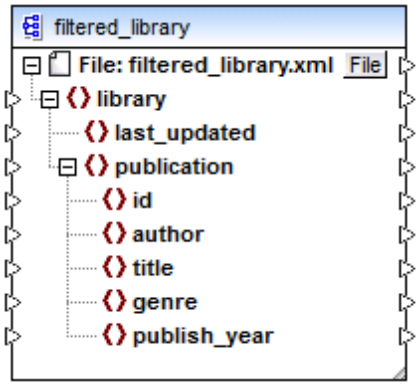


As shown above, the mapping now has two source components: **books** and **library**, and two target components. To distinguish between the target components, we will rename the second one to **filtered_library**, and also set the name of the XML file that should be generated by it. To do this, double-click the header of the right-most component and edit the component settings as follows:



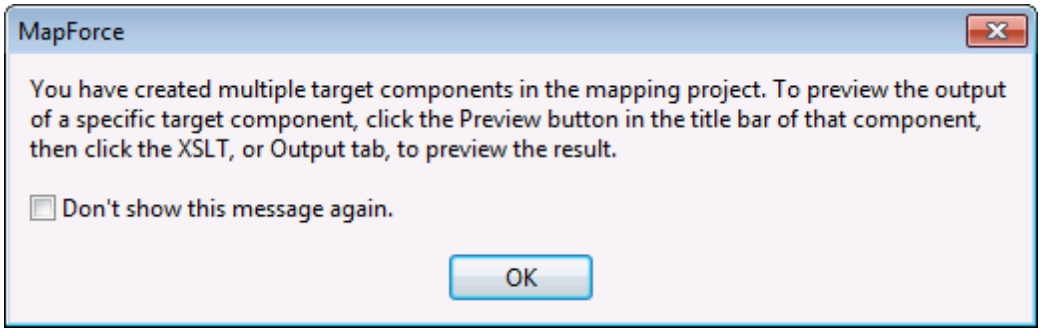
Notice that the new name of the component is **filtered_library**, and the output XML file is named **filtered_library.xml**.

 © 2015 Altova GmbH

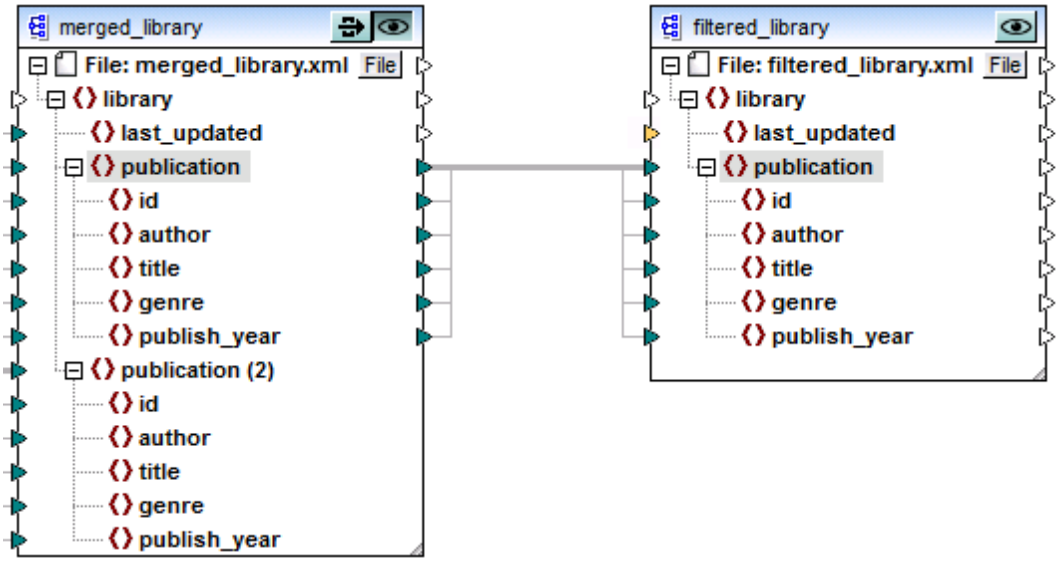


Step 3: Make the connections

Create a connection from the item **publication** in the **merged_library** to the item **publication** in the **filtered_library**. When you do this, a notification message is displayed.


MapForce
You have created multiple target components in the mapping project. To preview the output of a specific target component, click the Preview button in the title bar of that component, then click the XSLT, or Output tab, to preview the result.
Don't show this message again.
OK

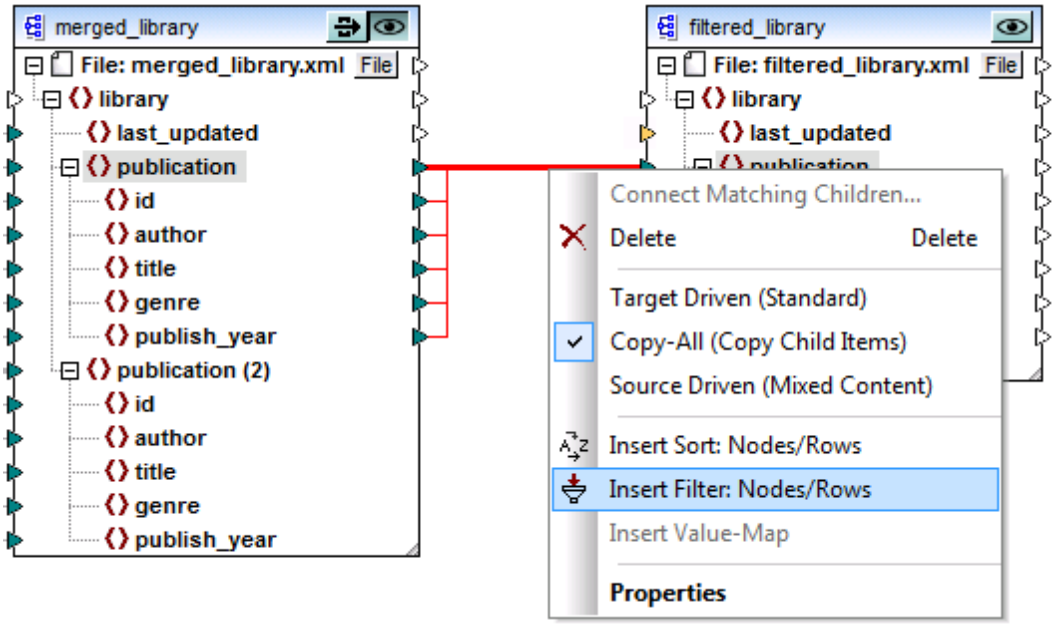
Click **OK**. Notice that new buttons are now available in the upper-right corner of both target components: **Preview** ( ) and **Pass-through** ( ). These buttons will be used and explained in the following steps.



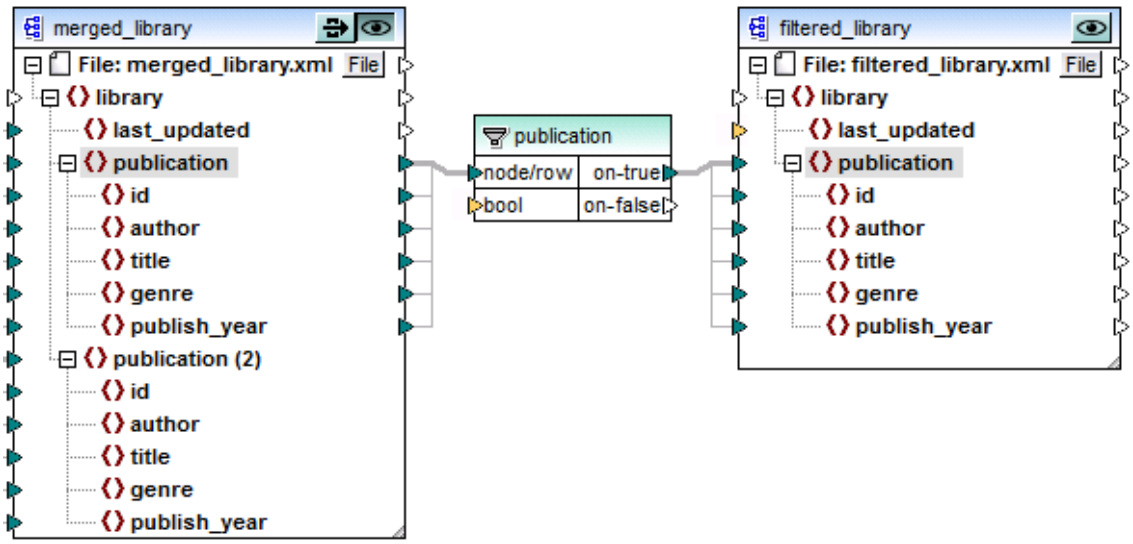
Step 4: Filter data

To filter data before supplying it to the **filtered_library**, we will use a **Filter** component. To add a filter component, right-click the connection between **merged_library** and **filtered_library**, and select **Insert Filter: Nodes/Rows** from the context menu.

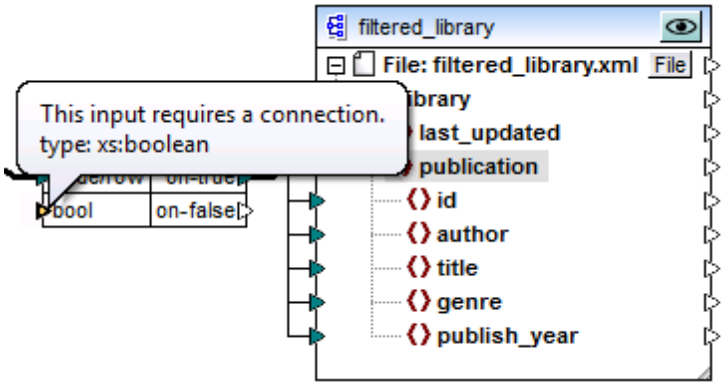
© 2015 Altova GmbH



The filter component has now been added to the mapping.



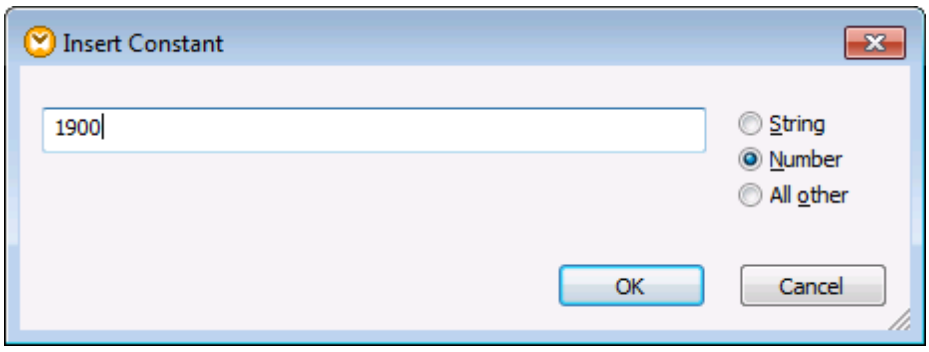
As shown above, the bool input connector is highlighted in orange, which suggests that an input is required. If you move the mouse over the connector, you can see that an input of type `xs:boolean` is required. Note that, for tips to be displayed, the **Show tips** ( ) toolbar button must be enabled.



The filter component requires a condition that returns either `true` or `false`. When the Boolean condition returns `true`, data of the current **publication** sequence will be copied over to the target. When the condition returns `false`, data will not be copied.

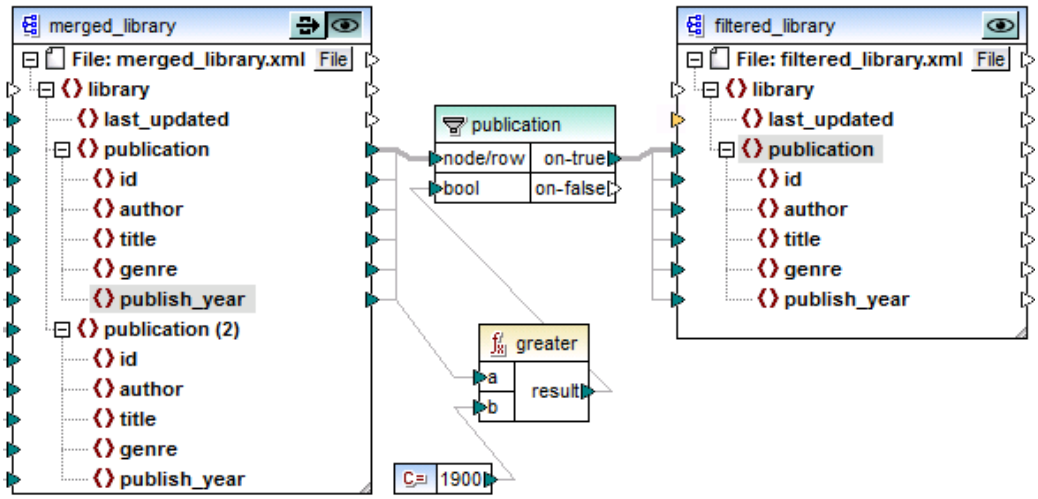
In this tutorial, the required condition is to filter all books which were published after 1900. To create the condition, do the following:

1. Add a constant of numeric type having the value "1900" (On the **Insert** menu, click **Constant**). Choose **Number** as type.



2. In the Libraries window, locate the function **`greater`** and drag it to the mapping pane.
3. Make the mapping connections to and from the function **`greater`** as shown below. By doing this, you are instructing MapForce: "When `publish_year` is greater than 1900, copy the current `publication` source sequence to the `publication` target sequence".

 © 2015 Altova GmbH



Step 5: Preview and save the output of each target component

You are now ready to preview and save the output of both target components. When multiple target components exist in the same mapping, you can choose which one to preview by clicking the **Preview** ( ) button. When the **Preview** button is in a pressed state ( ), it indicates that that specific component is currently enabled for preview (and this particular component will generate the output in the Preview pane). Only one component at a time can have the preview enabled.

Therefore, when you want to view and save the output of the **merged_library** (that is, the "intermediate") component, do the following:

1. Click the **Preview** button ( ) on the **merged_library** component.
2. Click the **Output** button at the bottom of the mapping pane.
3. On the **Output** menu, click **Save Output File** if you want to save the output to a file.

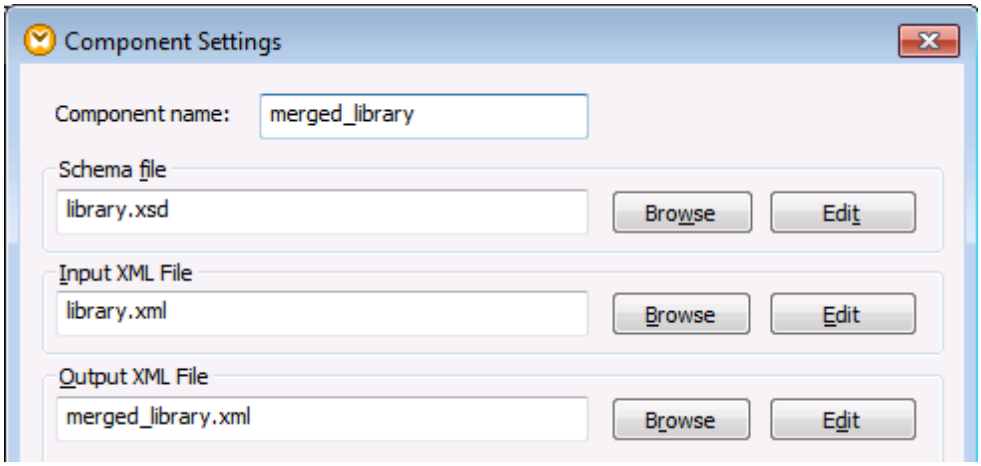
When you want to view and save the output of the **filtered_library** component :

1. Click the **Pass-through** button ( ) on the **merged_lilbrary** component.
2. Click the **Preview** button ( ) on the **filtered_library** component.
3. Click the **Output** button at the bottom of the mapping pane.
4. On the **Output** menu, click **Save Output FIle** if you want to save the output to a file.

Notice the **Pass-through** ( ) button—clicking or not clicking it makes a big difference in any mapping which has multiple target components, including this one. When this button is in a pressed state ( ), MapForce lets data pass through the intermediate component, so that you can preview the result of the entire mapping.

Release the button ( ) if you want to preview only the portion of the mapping between the **merged_library** and the **filtered_library**. In the latter case, an error will be generated. This behavior is expected, because the intermediate component does not have a valid input XML file from which it should read data. To solve the problem, double-click the header of the component

and edit so as to supply a valid input XML file, as shown below:



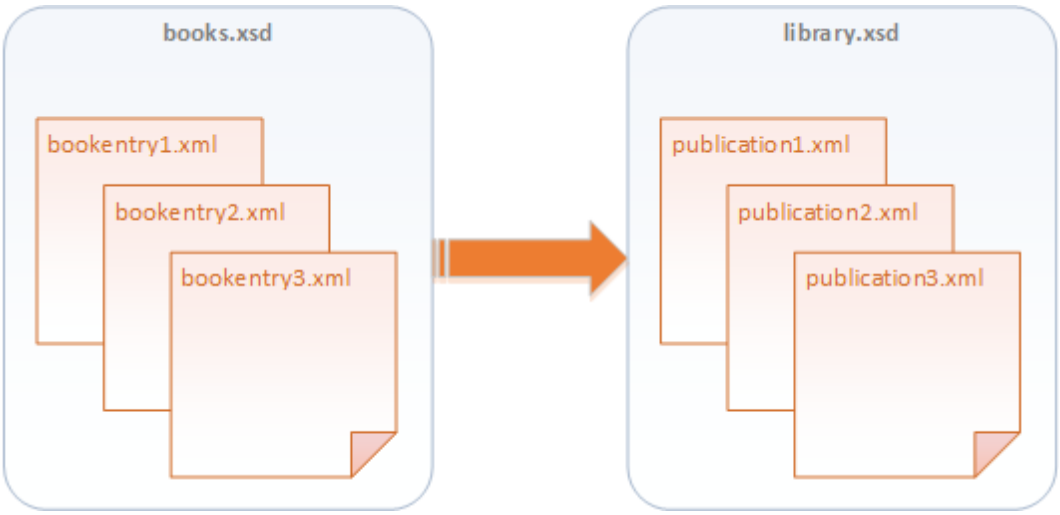
You have now finished designing a mapping which has multiple target components, and you can view and save the output of each target, which was the intended goal of this tutorial. For further information about working with pass-through components, see Chained mappings / pass-through components.

© 2015 Altova GmbH

3.4 Process and Generate Files Dynamically

This tutorial shows you how to read data from multiple source XML files and write it to multiple target files in the same transformation. To illustrate this technique, we will now create a mapping with the following goals:

1. Read data from multiple XML files in the same directory.
2. Convert each file to a new XML schema.
3. For each source XML file, generate a new XML target file under the new schema.
4. Strip the XML and namespace declaration from the generated files.



Abstract model of the data transformation

We will use three source XML files as example. The files are located in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder, and they are named **bookentry1.xml**, **bookentry2.xml**, and **bookentry3.xml**. Each of the three files stores a single book.

```
<?xml version="1.0" encoding="UTF-8"?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="books.xsd">
   <book id="1">
      <author>Mark Twain</author>
      <title>The Adventures of Tom Sawyer</title>
      <category>Fiction</category>
      <year>1876</year>
   </book>
</books>
```

bookentry1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="books.xsd">
   <book id="2">
      <author>Franz Kafka</author>
      <title>The Metamorphosis</title>
      <category>Fiction</category>
      <year>1912</year>
   </book>
</books>
```

bookentry2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="books.xsd">
   <book id="3">
      <author>Herman Melville</author>
      <title>Moby Dick</title>
      <category>Fiction</category>
      <year>1851</year>
   </book>
</books>
```

bookentry3.xml

The source XML files use the **books.xsd** schema available in the following folder: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial**. To convert the source files to a new XML schema, we will use the **library.xsd** schema (available in the same folder). After the transformation, the mapping will generate three files according to this new schema (see the code listings below). We will also configure the mapping so that the name of the generated files will be: **publication1.xml**, **publication2.xml**, and **publication3.xml**. Notice that the XML declaration and the namespace declaration must be stripped.

```
<library>
   <publication>
      <id>1</id>
      <author>Mark Twain</author>
      <title>The Adventures of Tom Sawyer</title>
      <genre>Fiction</genre>
      <publish_year>1876</publish_year>
   </publication>
</library>
```

publication1.xml

© 2015 Altova GmbH

```
<library>
   <publication>
      <id>2</id>
      <author>Franz Kafka</author>
      <title>The Metamorphosis</title>
      <genre>Fiction</genre>
      <publish_year>1912</publish_year>
   </publication>
</library>
```

publication2.xml

```
<library>
   <publication>
      <id>3</id>
      <author>Herman Melville</author>
      <title>Moby Dick</title>
      <genre>Fiction</genre>
      <publish_year>1851</publish_year>
   </publication>
</library>
```

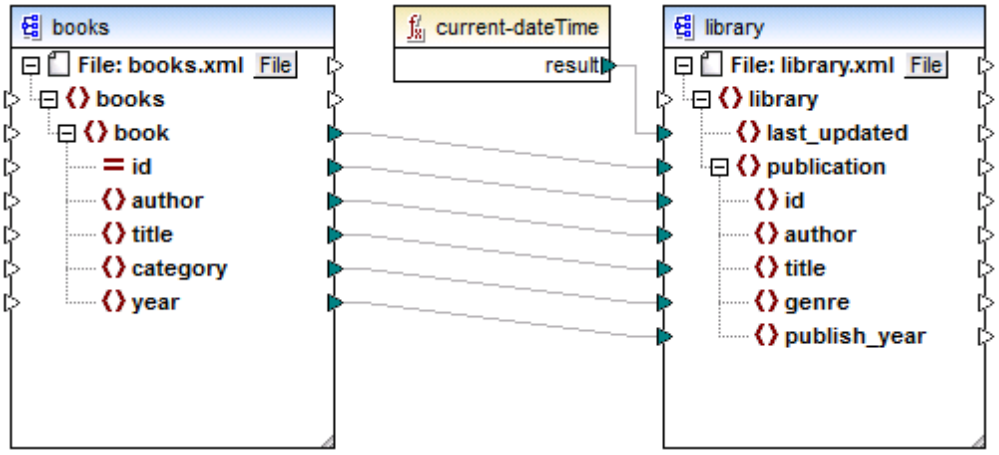
publication3.xml

To achieve the goals, let's take the following steps.

Step 1: Prepare the mapping design file

This tutorial uses as starting point the **BooksToLibrary.mfd** mapping from the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder. You have already designed this mapping in the Convert XML to New Schema tutorial. To begin, open the **BooksToLibrary.mfd** file in MapForce, and save it with a new name, in the same folder.

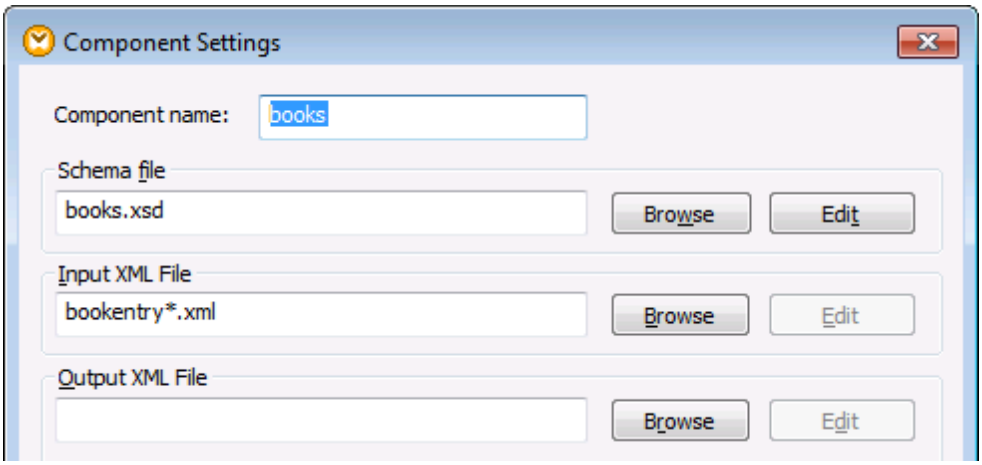
> Make sure to save the new mapping in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder, because it references several files from it.



BooksToLibrary.mfd (MapForce Basic Edition)

Step 2: Configure the input

To instruct MapForce to process multiple XML instance files, double-click the header of the source component. In the Component Settings dialog box, enter **bookentry*.xml** as input file.



Component Settings dialog box

The asterisk (*) wildcard character in the file name instructs MapForce to use as mapping input all the files that have the **bookentry-** prefix. Because the path is a relative one, MapForce will look for all **bookentry-** files in the same directory as the mapping file. Note that you could also enter an absolute path if necessary, while still using the * wildcard character.

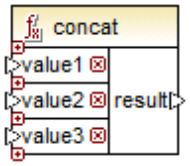
Step 3: Configure the output

To create the file name of each output file, we will use the **concat** function. This function concatenates (joins) all the values supplied to it as argument.

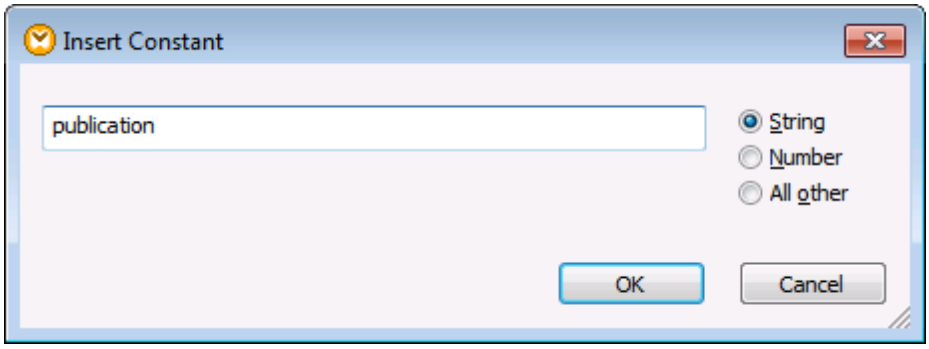
 © 2015 Altova GmbH

To build the file name using the **concat** function:

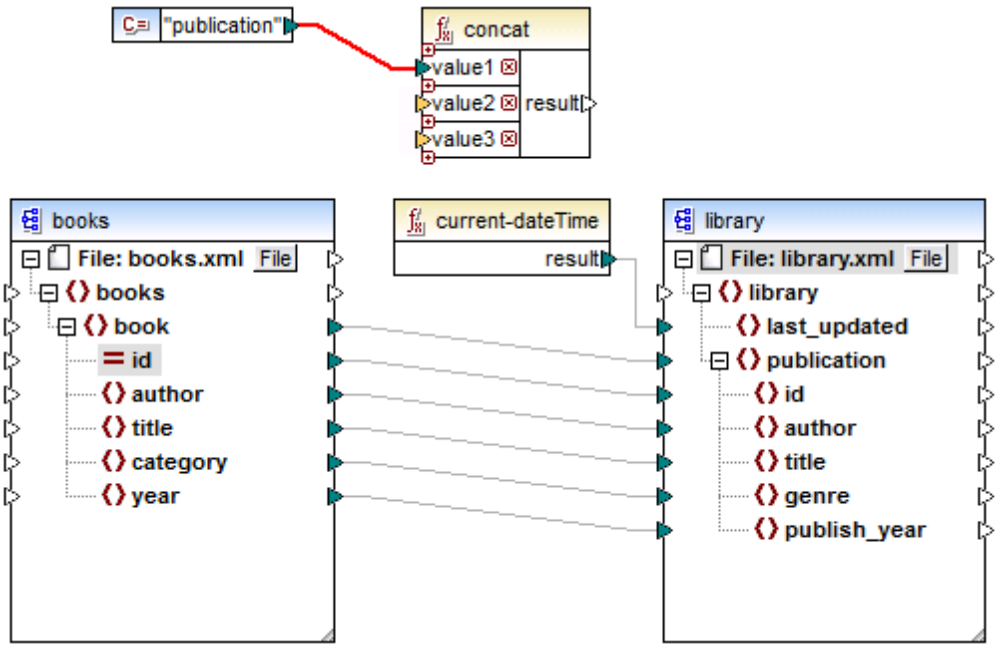
1. Search for the **concat** function in the Libraries window and drag it to the mapping area. By default, this function is added to the mapping with two parameters; however, you can add new parameters if necessary. Click the **Add parameter** (⊡) symbol inside the function component and add a third parameter to it. Note that clicking the **Delete parameter** (⊠) symbol deletes a parameter.



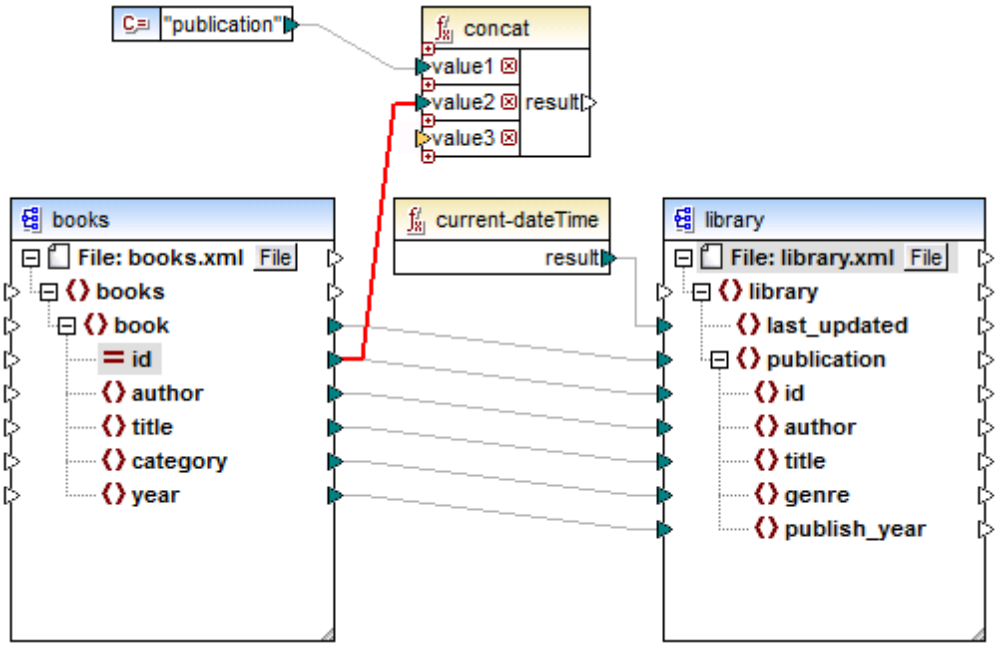
2. Insert a constant (on the **Insert** menu, click **Constant**). When prompted to supply a value, enter "publication" and leave the **String** option unchanged.



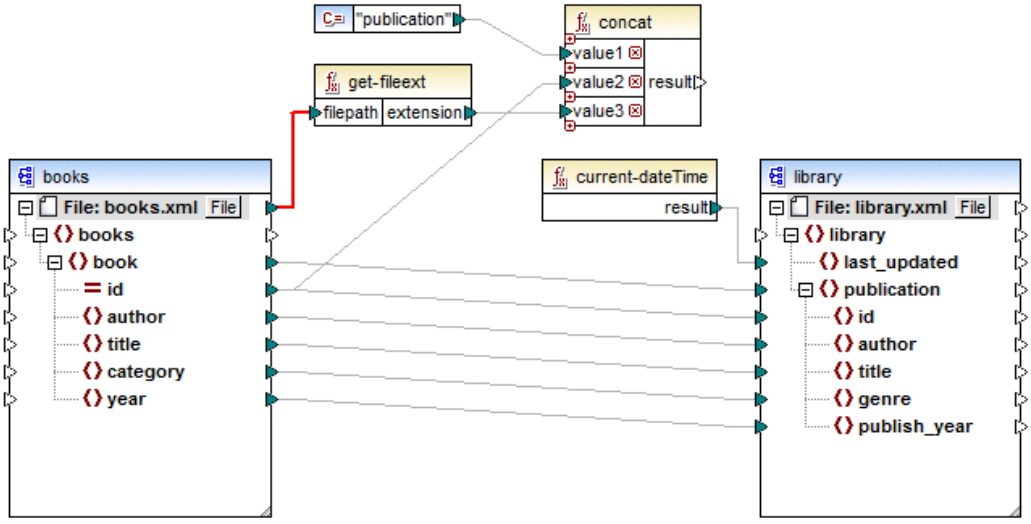
3. Connect the constant with **value1** of the **concat** function.



4. Connect the `id` attribute of the source component with **value2** of the concat function.



5. Search for the **get-fileext** function in the Libraries window and drag it to the mapping area. Create a connection from the top node of the source component (**File: books.xml**) to the **filepath** parameter of this function. Then create a connection from the result of the **get-fileext** function to **value3** of the **concat** function. By doing this, you are extracting only the extension part (in this case, .xml) from the source file name.

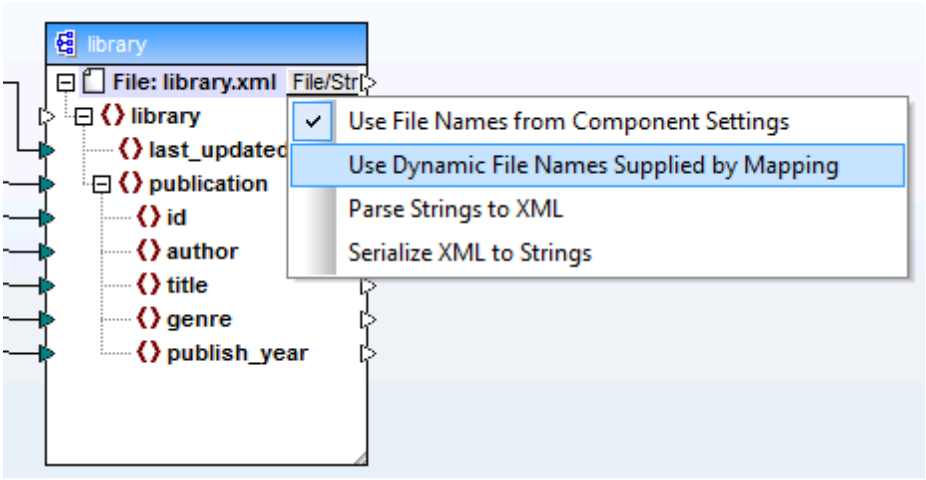


So far, you have provided as parameters to the **concat** function the three values which, when joined together, will create the generated file name (for example, **publication1.xml**):

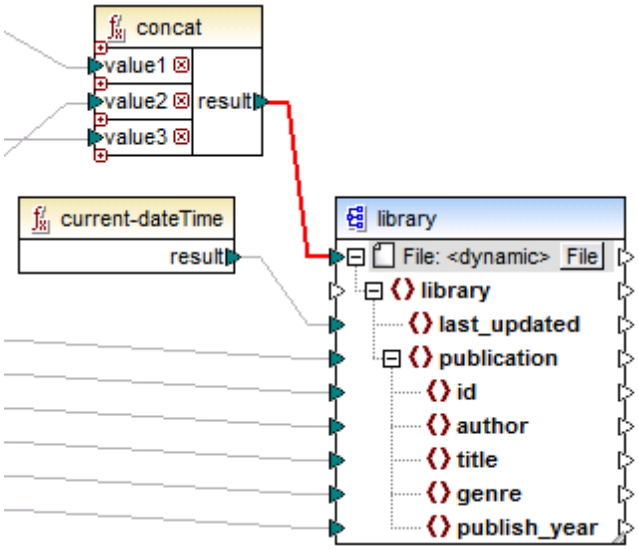
 © 2015 Altova GmbH

Part	Example
The constant "publication" supplies the constant string value "publication".	publication
The attribute `id` of the source XML file supplies a unique identifier value for each file. This is to prevent all files from being generated with the same name.	1
The **`get-fileext`** function returns the extension of the file name to be generated.	.xml

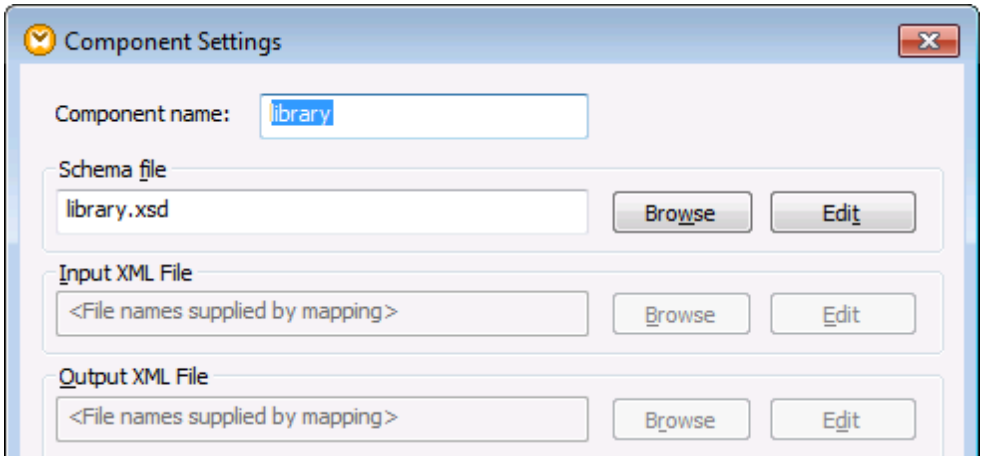
You can now instruct MapForce to actually build the file name when the mapping runs. To do this, click the **File** (File) or **File/String** (File/String) button of the target component and select **Use Dynamic File Names Supplied by Mapping**.



You have now instructed MapForce to generate the instance files dynamically, with whatever name will be provided by the mapping. In this particular example, the name is created by the **`concat`** function; therefore, we will connect the result of the **`concat`** function with the **File: <dynamic>** node of the target component.



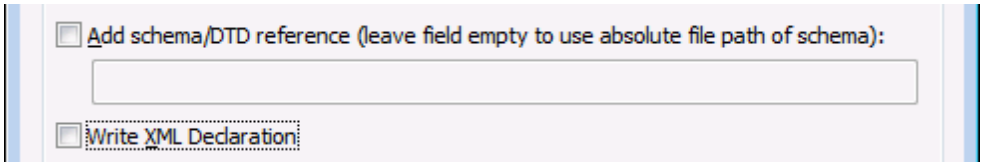
If you double-click the target component header at this time, you will notice that the **Input XML File** and **Output XML File** text boxes are disabled, and their value shows **<File names supplied by the mapping>**.



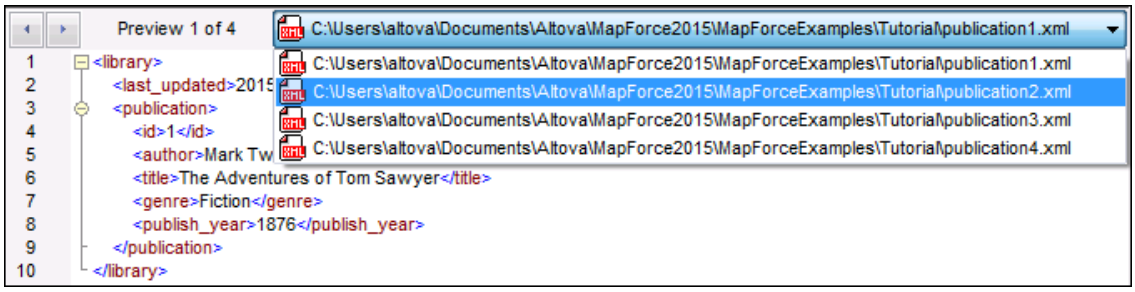
This serves as an indication that you have supplied the instance file names dynamically from a mapping, so it is no longer relevant to define them in the component settings.

Finally, you need to strip the XML namespace and schema declaration from the target. To achieve this, clear the selection from the **Add schema/DTD reference...** and **Write XML Declaration** check boxes on the Component Settings dialog box.

 © 2015 Altova GmbH



You can now run the mapping and see the result, as well as the name of generated files. This mapping generates multiple output files. You can navigate through the output files using the left and right buttons in the upper left corner of the output pane, or by picking a file from the adjacent drop-down list.



Chapter 4

Common Tasks

4 Common Tasks

This section describes common MapForce tasks and concepts, such as working with mappings, components, connections, as well as using relative and absolute paths.

© 2015 Altova GmbH

4.1 Working with Mappings

A MapForce mapping design (or simply "mapping") is the visual representation of how data is to be transformed from one format to another. A mapping consists of components that you add to the MapForce mapping area in order to create your data transformations (for example, convert XML documents from one schema to another). A valid mapping consists of one or several source components connected to one or several target components. You can run a mapping and preview its result directly in MapForce. You can generate code and execute it externally. You can also compile a mapping to a MapForce execution file and automate mapping execution using MapForce Server or FlowForce Server. MapForce saves mappings as files with .mfd extension.

To create a new mapping:

1. Do one of the following:
   - On the **File** menu, click **New**.
   - Click the **New** ( ) toolbar button.

Your mapping is now created; however, it does not yet do anything because it is empty. A mapping requires at least two connected components to become valid, so the next step is to add components to the mapping (see Adding Components to the Mapping) and draw connections between components (see Working with Connections).

4.1.1 Adding Components to the Mapping

In MapForce, the term "component" is what represents visually the structure (schema) of your data, or how data is to be transformed (functions). Components are the central building pieces of any mapping. On the mapping area, components appear as rectangles. The following are examples of MapForce components:

- Constants
- Filters
- Conditions
- Function components
- EDI documents (UN/EDIFACT, ANSI X12, HL7)
- Excel 2007+ files
- Simple input components
- Simple output components
- XML Schemas and DTDs

To add a component to the mapping, do one of the following:

- On the **Insert** menu, click the option relevant for the component type you wish to add (for example, **XML Schema/File**).
- Drag a file from Windows explorer onto the mapping area. Note that this operation is possible only for compatible file-based components.
- Click the relevant button on the Insert Component toolbar.

Insert Component toolbar (MapForce Enterprise Edition)

Each component type has specific purpose and behavior. For component types where that is necessary, MapForce walks you through the process by displaying contextual wizard steps or dialog boxes. For example, if you are adding an XML schema, a notification dialog box prompts you to optionally select an instance file as well.

For an introduction to components, see Working with Components. For specific information about each technology supported as mapping source or target, see Data Sources and Targets. For information about MapForce built-in components used to store data temporarily or transform it (such as filtering or sorting), see Designing Mappings.

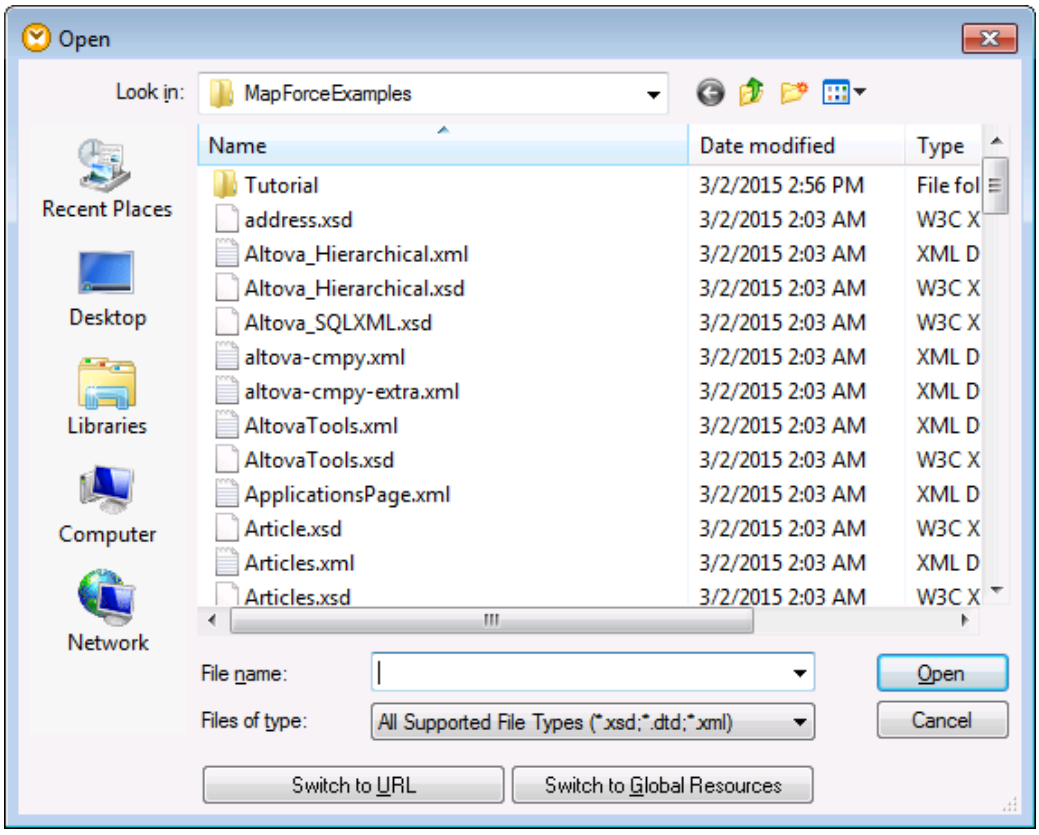
4.1.2 Adding Components from a URL

In addition to adding local files as mapping components, you can also add files from a URL. Note that this operation is supported when you add a component as source component (that is, your mapping reads data from the remote file).

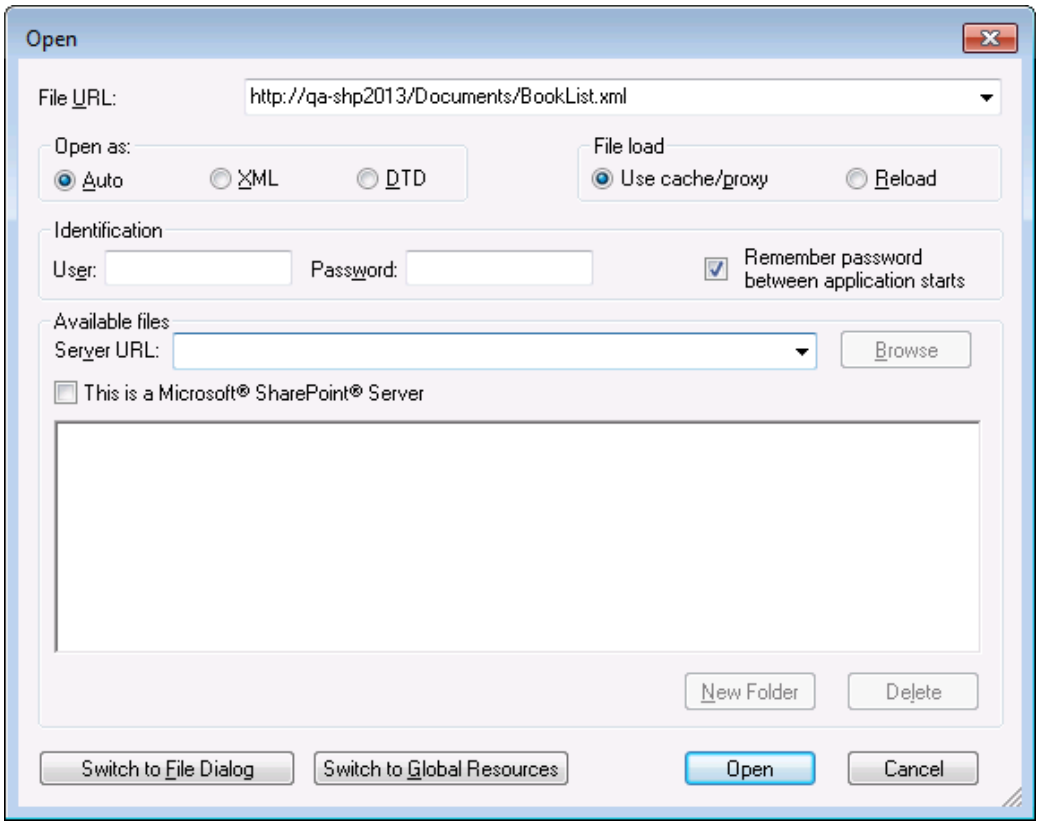
To add a component from a URL:

1. On the **Insert** menu, select the type of the component type you wish to add (for example, **XML Schema/File**).
2. On the **Open** dialog box, click **Switch to URL**.

© 2015 Altova GmbH



3. Enter the URL of the file in the **File URL** text box, and click **Open**.



Make sure that the file type in the **File URL** text box is the same as the file type you specified in step 1.

If the server requires password authentication, you will be prompted to enter the user name and password. If you want the user name and password to be remembered next time you start MapForce, enter them in the Open dialog box and select the **Remember password between application starts** check box.

The **Open As** setting defines the grammar for the parser when opening the file. The default and recommended option is **Auto**.

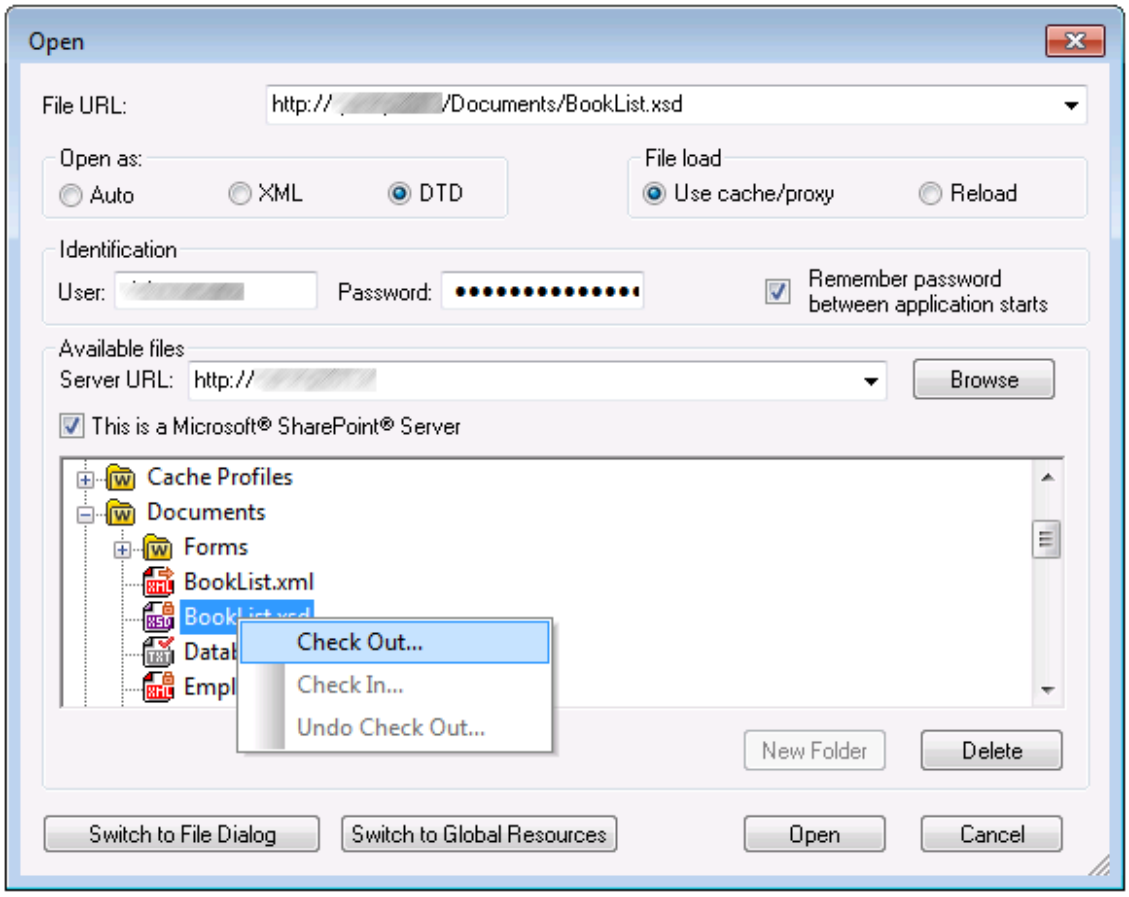
If the file you are loading is not likely to change, select the **Use cache/proxy** option to cache data and speed up loading the file. Otherwise, if you want the file to be reloaded each time when you open the mapping, select **Reload**.

For servers with Web Distributed Authoring and Versioning (WebDAV) support, you can browse files after entering the server URL in the **Server URL** text box and clicking **Browse**. Although the preview shows all file types, make sure that you choose to open the same file type as specified in step 1 above; otherwise, errors will occur.

If the server is a Microsoft SharePoint Server, select the **This is a Microsoft SharePoint Server** check box. Doing so displays the check-in or check-out state of the file in the preview area. If you want to make sure that no one else can edit the file on the server while you are using it in

 © 2015 Altova GmbH

MapForce to read data from it, right-click the file and select **Check Out**. To check in any file that was previously checked out by you, right-click the file and select **Check In**.



Open dialog box (in Switch to URL mode)

4.1.3 Selecting a Transformation Language

You can choose one of the following as data transformation language:

- XSLT 1.0
- XSLT 2.0

To select a transformation language, do one of the following:

- On the **Output** menu, click the name of the language you wish to use for transformation.
- Click the name of the language in the Language Selection toolbar.

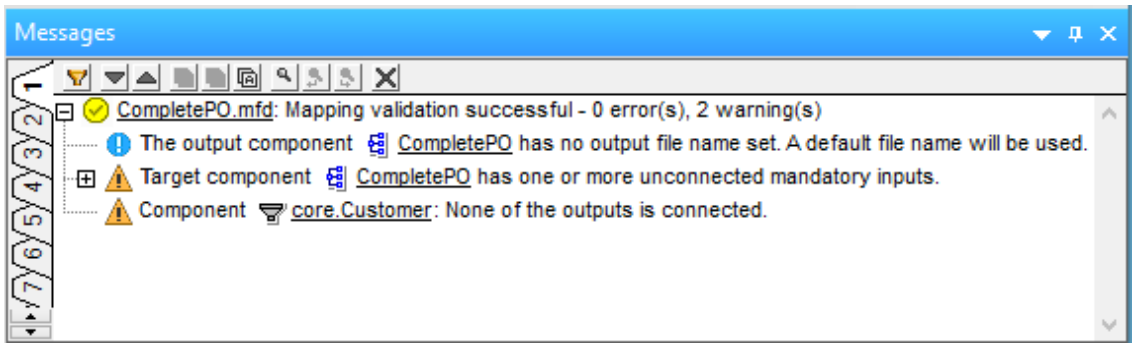
4.1.4 Validating Mappings

MapForce validates mappings automatically, when you click the **Output** tab to preview the transformation result. You can also validate a mapping explicitly, before attempting to preview its result. This helps you identify and correct potential mapping errors and warnings before the mapping is run. Note that running a mapping may generate additional runtime errors or warnings depending on the processed data, for example, when values mapped to attributes are overwritten.

To validate a mapping explicitly, do one of the following:

- On the **File** menu, click **Validate Mapping**.
- Click the **Validate** ( ) toolbar button.

The Messages window displays the validation results, for example:


Messages
CompletePO.mfd: Mapping validation successful - 0 error(s), 2 warning(s)
The output component CompletePO has no output file name set. A default file name will be used.
Target component CompletePO has one or more unconnected mandatory inputs.
Component core.Customer: None of the outputs is connected.

Messages window

When you validate a mapping, MapForce checks for the validity of the mapping (such as incorrect or missing connections, unsupported component kinds), and the validation result is then displayed in the Messages window with one of the following status icons:

Icon	Meaning
	Validation has completed successfully.
	Validation has completed with warnings.
	Validation has failed.

The Message window may additionally display any of the following message types: information messages, warnings, and errors.

Icon	Meaning
	Denotes an information message. Information messages do not stop the mapping execution.
	Denotes a warning message. Warnings do not stop the mapping execution.

© 2015 Altova GmbH

Icon	Meaning
	They may be generated, for example, when you do not create connections to some mandatory input connectors. In such cases, output will still be generated for those component where valid connections exist.
	Denotes an error. When an error occurs, the mapping execution fails, and no output is generated. The preview of the XSLT or XQuery code is also not possible.

To highlight on the mapping area the component or structure which triggered the information, warning, or error message, click the underlined text in the Messages window.

For components that transform data (such as functions or variables), MapForce validation works as follows:

- If a mandatory **input connector** is unconnected, an error message is generated and the transformation is stopped.
- If an **output connector** is unconnected, then a warning is generated and the transformation process continues. The offending component and its data are ignored and are not mapped to the target document.

To display the result of each validation in an individual tab, click the numbered tabs available on the left side of the Messages window. This may be useful, for example, if you work with multiple mapping files simultaneously

Other buttons in the Messages window enable you to take the following actions:

- Filter the message by types (for example, to show only errors or warnings)
- Move up or down through the entries
- Copy the message text to the clipboard
- Find a specific text in the window
- Clear the Messages window.

4.1.5 Validating the Mapping Output

After you click the **Output** tab to preview the mapping, the resulting output becomes available in the Output pane. You can validate this output against the schema associated with it. For example, if the mapping transformation generates an XML file, then the resulting XML document can be validated against the XML schema.

For XML files, you can specify the schema associated with the instance file in the **Add Schema/ DTD reference** field of the Component Settings dialog box (see XML Component Settings). The path specifies where the schema file referenced by the produced XML output is to be located. This ensures that the output instance can be validated when the mapping is executed. You can enter an `http://` address in this field, as well as an absolute or relative path. If you do not select the **Add Schema/DTD reference** field, then the validation of the output file against the schema is not possible. If you select this check box but leave it empty, then the schema filename of the Component Settings dialog box is generated into the output and the validation is done against it.

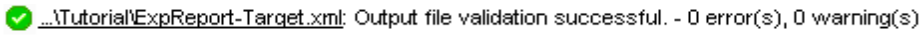
To validate the mapping output, do one of the following:

- Click the **Validate** toolbar button.

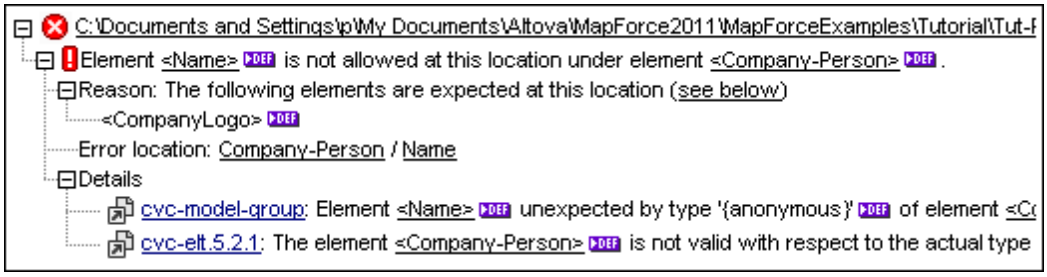
- On the **Output** menu, click **Validate Output File**.

Note: The **Validate** button and its corresponding menu command (**Output | Validate Output File**) are enabled only if the output file supports validation against a schema.

The result of the validation is displayed in the Messages window, for example:



If the validation was not successful, the message contains detailed information on the errors that occurred.



The validation message contains a number of hyperlinks you can click for more detailed information:

- Clicking the file path opens the output of the transformation in the **Output** tab of MapForce.
- Clicking <ElementName> link highlights the element in the **Output** tab.
- Clicking the icon opens the definition of the element in XMLSpy (if installed).
- Clicking the hyperlinks in the Details subsection (e.g., cvc-model-group) opens a description of the corresponding validation rule on the http://www.w3.org/ website.

4.1.6 Previewing the Output

When working with MapForce mappings, you can preview the resulting output without having to run and compile the generated code with an external processor or compiler. In general, it is a good idea to preview the transformation output within MapForce before attempting to process the generated code externally.

When you choose to preview the mapping results, MapForce executes the mapping and populates the Output pane with the resulting output. Once data is available in the Output pane, you can validate and save it if necessary. You can also use the **Find** command (**Ctrl + F** key combination) to quickly locate a particular text pattern within the output file.

To preview the transformation output:

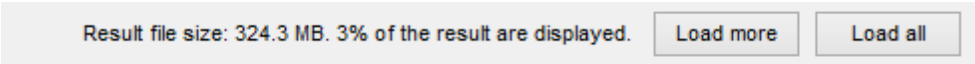
- Click the **Output** tab under the Mapping window. MapForce executes the mapping using the transformation language selected in the Language toolbar and populates the Output pane with the resulting output.

To save the transformation output, do one of the following:

- On the **Output** menu, click **Save Output File**.
- Click the **Save Generated Output** toolbar button.

Partial output preview

When you are previewing large output files, MapForce limits the amount of data displayed in the Output pane. More specifically, MapForce displays only a part of the file in the Output pane, and a **Load more...** button appears in the lower area of the pane. Clicking the **Load more...** button appends the next file part to the currently visible data, and so on.



Note: The **Pretty-print** button becomes active when the complete file has been loaded into the Output pane.

You can configure the preview settings from the **General** tab of the **Options** dialog box.

4.1.7 Previewing the XSLT Code

You can preview the XSLT code generated by MapForce if you selected XSLT 1.0 or XSLT 2.0 as data transformation language (see Selecting a transformation language).

To preview the generated XSLT 1.0 (or XSLT 2.0) code, do one of the following:

- To preview the XSLT 1.0 code, click the **XSLT** tab under the Mapping window.
- To preview the XSLT 2.0 code, click the **XSLT2** tab under the Mapping window.

Note: The XSLT (or XSLT2) tab becomes available if you have selected XSLT (or XSLT2, respectively) as transformation language.

4.1.8 Generating XSLT Code

To generate XSLT code:

1. Select the menu item **File | Generate code in | XSLT 1.0 (XSLT 2.0)**.
2. Select the folder you want to place the generated XSLT file in, and click **OK**. MapForce generates the code and displays the result of the operation in the Messages window.
3. Navigate to the designated folder and you will find the XSLT file name in the form: **MappingMapTo<TargetSchemaName>.xslt**.

The folder in which the XSLT file is placed also contains a batch file called **DoTransform.bat** which can be run with RaptorXML Server to transform the data.

To run the transformation with RaptorXML Server:

1. Download and install RaptorXML from the download page.
2. Start the **DoTransform.bat** batch file located in the previously designated output folder.

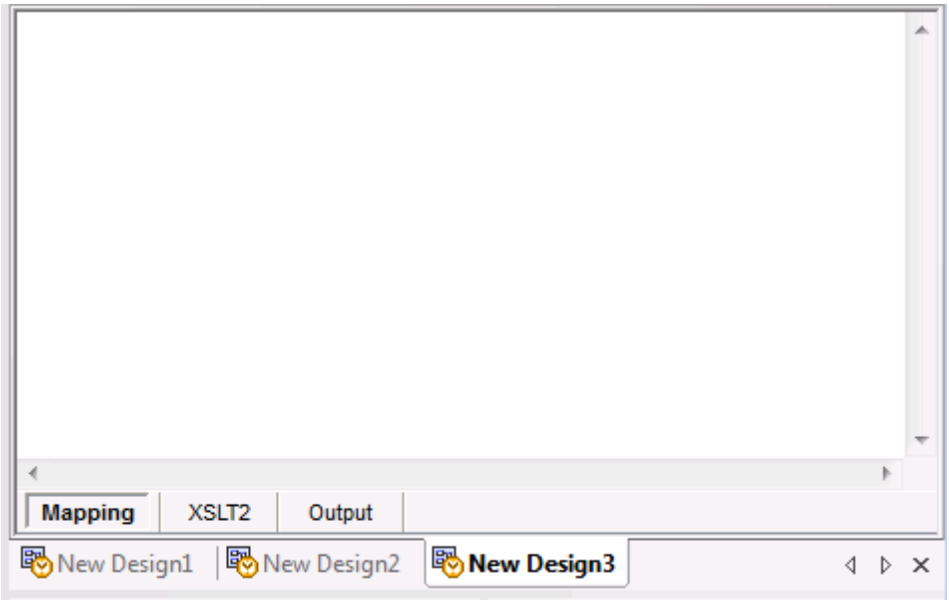
Note that you might need to add the RaptorXML installation location to the **path** variable of the Environment Variables. You can find the RaptorXML documentation on the website documentation page.

4.1.9 Working with Multiple Mapping Windows

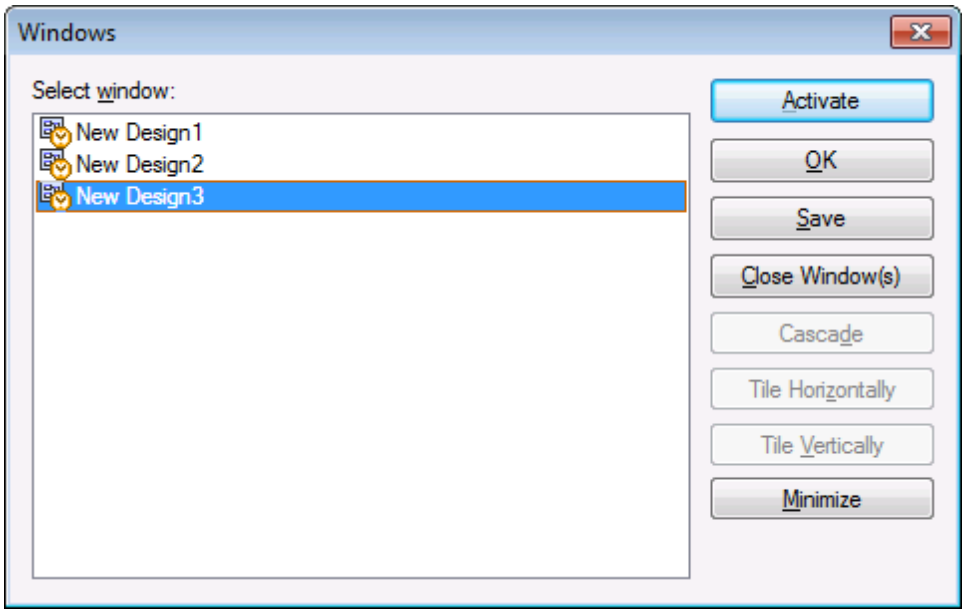
MapForce uses a Multiple Document Interface (MDI). Each mapping file you open in MapForce has a separate window. This enables you to work with multiple mapping windows and arrange or resize them in various ways inside the main (parent) MapForce window. You can also arrange all open windows using the standard Windows layouts: Tile horizontally, Tile vertically, Cascade.

When multiple mappings are open in MapForce, you can quickly switch between them using the tabs displayed in the lower part of the Mapping pane.

© 2015 Altova GmbH



Window management options are available both on the **Window** menu and on the **Windows** dialog box. From the **Windows** dialog box, you can take actions against any or all currently open mapping windows (including saving, closing, or minimizing them).



Windows dialog box

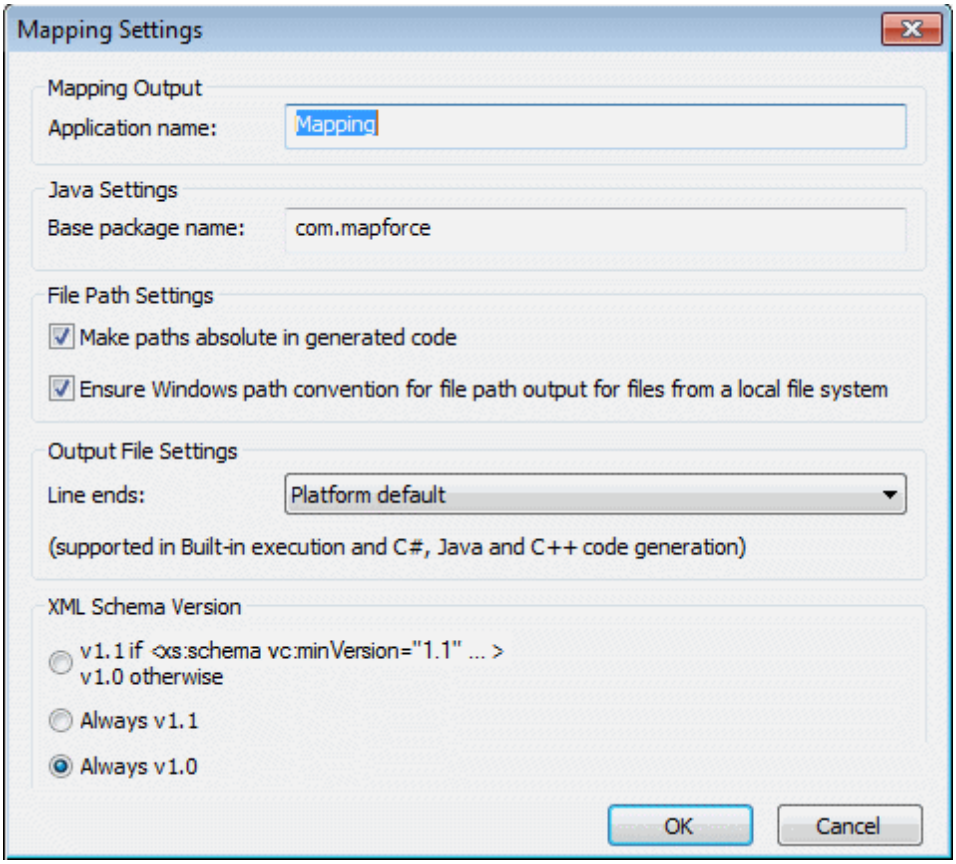
You can open the Windows dialog box using the menu command **Window | Windows...** To select multiple windows in the Windows dialog box, click the required entries while holding the **Ctrl** key pressed.

4.1.10 Changing the Mapping Settings

You can change the document-specific settings of the currently active mapping design file from the Mapping Settings dialog box. This information is stored in the *.mfd file.

To open the Mapping Settings dialog box:

- On the **File** menu, click **Mapping Settings**.

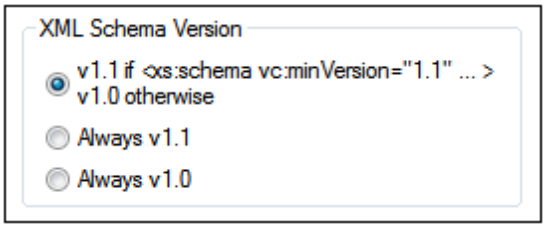


Mapping Settings dialog box

The available settings are as follows.

Application Name	Defines the XSLT1.0/2.0 file name prefix for the generated transformation files.
Make paths absolute in generated code	Defines whether the file paths should be relative or absolute in the generated code. For more information, see About Paths in Generated Code.
Ensure Windows path convention for file path	The "Ensure Windows path convention...." check box makes sure that Windows path conventions are followed. When outputting XSLT2 (and XQuery), the currently processed file

 © 2015 Altova GmbH

	name is internally retrieved using the document-uri function, which returns a path in the form file:// URI for local files. When this check box is active, a file:// URI path specification is automatically converted to a complete Windows file path (e.g. "C:\...") to simplify further processing.
XML Schema Version	Lets you define the XML Schema Version used in the mapping file. You can define if you always want to **load** the Schemas conforming to version 1.0 or 1.1. Note that not all version 1.1 specific features are currently supported. If the *xs:schema vc:minVersion="1.1"* declaration is present, then version 1.1 will be used; if not, version 1.0 will be used.  If the XSD document has no `vc:minVersion` attribute or the value of the `vc:minVersion` attribute is other than `1.0` or `1.1`, then XSD 1.0 will be the default mode. **Note:** Do not confuse the `vc:minVersion` attribute with the `xsd:version` attribute. The former holds the XSD version number, while the latter holds the document version number. Changing this setting in an existing mapping causes a reloading of all schemas of the selected XML schema version, and might also change its validity.

4.2 Working with Components

Components are the central elements of any mapping design in MapForce. Generally, the term "component" is a convenient way to call any object which acts as a data source, or as a data target, or represents your data in the mapping at an intermediary processing stage.

There are two main categories of components: structure components and transformation components.

The structure components represent the abstract structure or schema of your data. For example, when you add an XML file to the mapping area (using the menu command **Insert | XML Schema/ File**), it becomes a mapping component. For further information about structure components and their specifics, see Data Sources and Targets. With a few exceptions, structure components consist of items and sequences. An item is the lowest level mapping unit (for example, a single attribute in the XML file, or an element of simple type). A sequence is a collection of items.

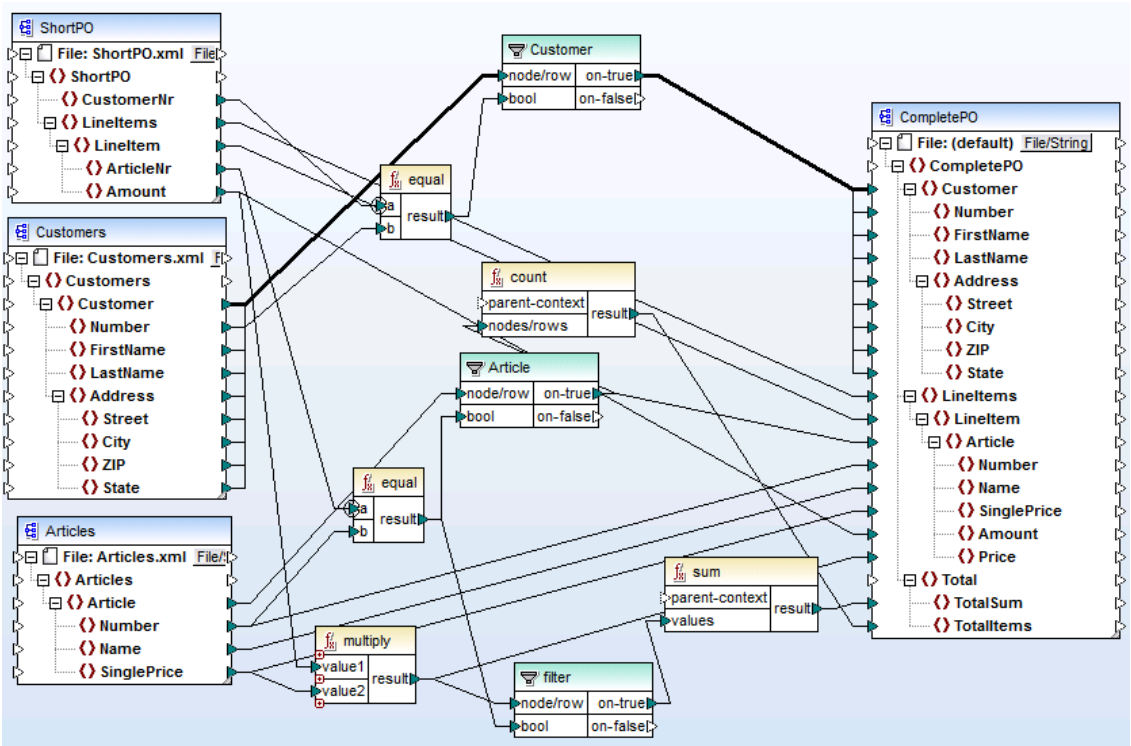
The transformation components either transform data (for example, functions), or assist you in transformations (for example, constants or variables). For information on how you can use these components to achieve various data transformation tasks, see Designing Mappings.

With the help of structure components, you can either read data from files or other sources, write data to files or other sources, or store data at some intermediary stage in the mapping process (for example, in order to preview it). Consequently, structure components can be of the following types:

- Source. You declare a component as source by placing it on the left of the mapping area, and, thus, instructing MapForce to read data from it.
- Target. You declare a component as target by placing on the right of the mapping area, and, thus, instructing MapForce to write data to it.
- Pass-through. This is a special component type which acts both as a source and target (for further information, see Chained mappings / pass-through components).

On the mapping area, components appear as rectangles. The following sample mapping illustrates three source components, one target XML component, and various transformation components (functions and filters) through which data goes before being written to the source.

© 2015 Altova GmbH



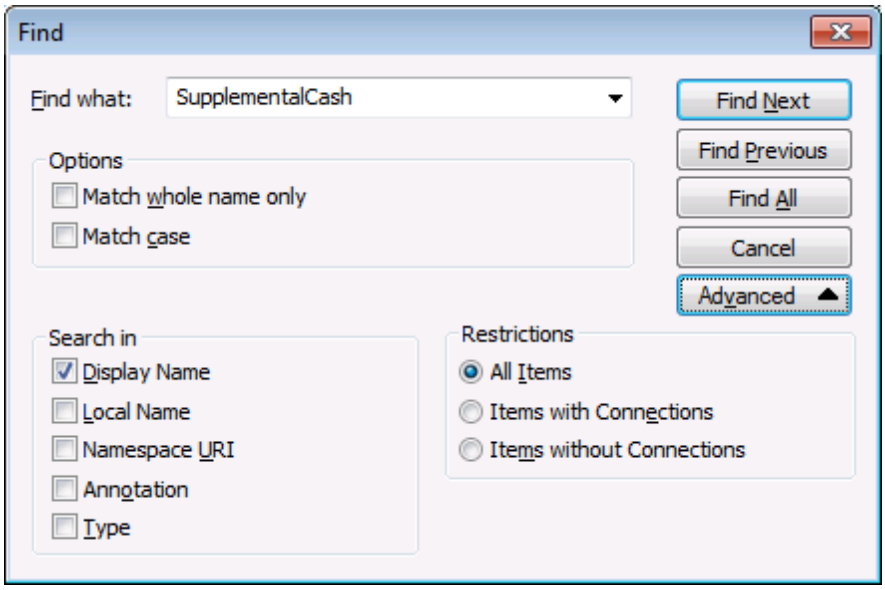
CompletePO.mfd

This mapping sample is available at the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\CompletePO.mfd**.

4.2.1 Searching within Components

To search for a specific node/item in a component:

1. Click the component you want to search in, and press the CTRL+F keys.
2. Enter the search term and click **Find Next**.

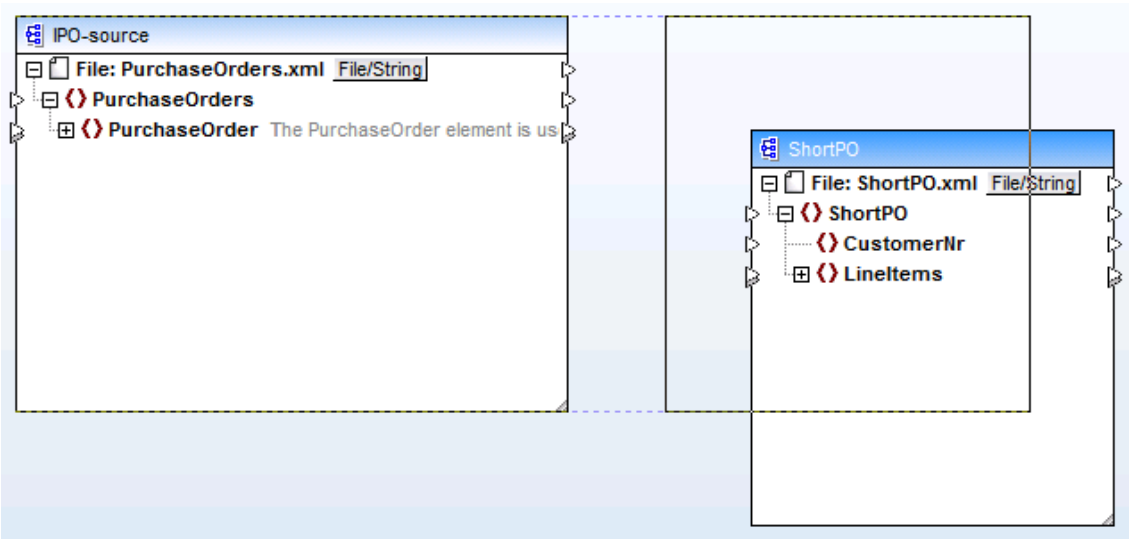


Use the Advanced options to define which items (nodes) are to be searched, as well as restrict the search options based on the specific connections.

4.2.2 Aligning Components

When you move components in the mapping pane, MapForce displays auto-alignment guide lines. These guide lines help you align a component to any other component in the mapping window.

In the sample mapping below, the lower component is being moved. The guide lines show that it can be aligned to the component on the left side of the mapping.



Component auto-alignment guide lines

 © 2015 Altova GmbH

To enable or disable this option:

1. On the **Tools** menu, click **Options**.
2. In the **Editing** group, select the **Align components on mouse dragging** check box.

4.2.3 Changing the Component Settings

After you add a component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:

- Select the component and, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.

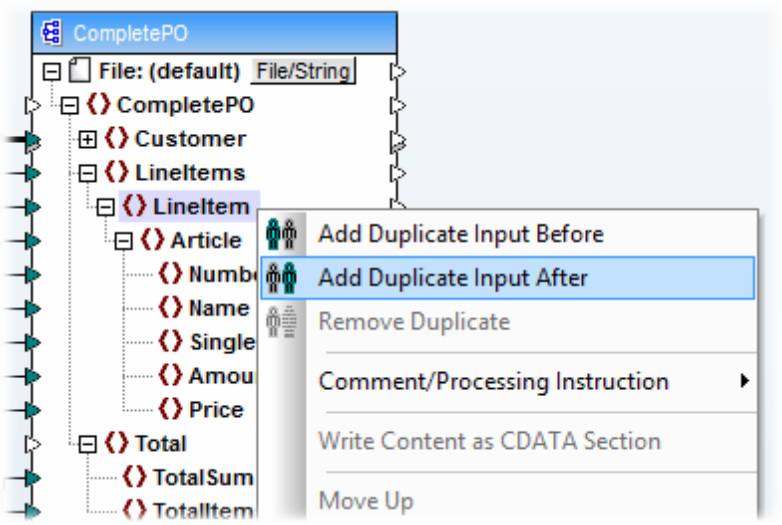
For a description of the settings available on the Component Settings dialog box, see XML Component Settings.

For any file-based component, such as XML, a **File** (File) button appears next to the root node. This button specifies advanced options applicable if you want to process or generate multiple files in a single mapping (see Processing Multiple Input or Output Files Dynamically).

4.2.4 Duplicating Input

Sometimes, you may need to configure a component to accept data from more than one source. For example, you may need to convert data from two different XML schemas into a single schema. To make the destination schema accept data from both source schemas, you can duplicate any of the input items in the component. Duplicating input is meaningful only for a component which is a target component. On any given target component, you can duplicate as many items as required.

To duplicate a particular input item, right-click it and select **Add Duplicate Input After/Before** from the context menu.



In the image above, the item `LineItem` is being duplicated in order to provide the ability to map data from a second source.

Once you duplicate an input, you can make connections both to the original input and to the duplicate input. For example, this would enable you to copy data from source A to original input, and data from source B to the duplicate input.

For a step-by-step example, see Map Multiple Sources to One Target.

© 2015 Altova GmbH

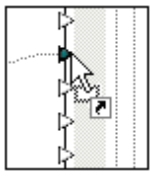
4.3 Working with Connections

A mapping is ultimately about transforming data from one format or structure into another. In a very basic mapping scenario, you add to the mapping area the components which represent your source and your target data (for example, a source XML schema and a destination one), and then draw visually the mapping connections between the two structure. A connection is, therefore, the visual representation of how data is mapped from a source to a destination.

Components have inputs and outputs which appear on the mapping as small triangles, called connectors. Input connectors are positioned to the left of any item to which you can draw a connection. Output connectors are positioned to the right of any item from which you can draw a connection.

To draw a connection between two items:

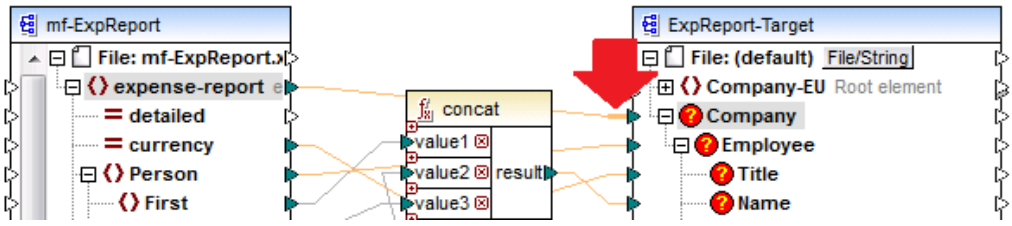
- Click the output connector of a source item and drag it to a destination item. When the drop action is allowed, a link tooltip appears next to the text cursor.



An input connector accepts only one incoming connection. If you try to add a second connection to the same input, a prompt appears asking if you want to replace the connection with a new one or duplicate the input item. An output connector can have several connections, each to a different input.

To move a connection to a different item:

- Click the stub of the connection (the straight section closer to the target) and drag it to the destination.

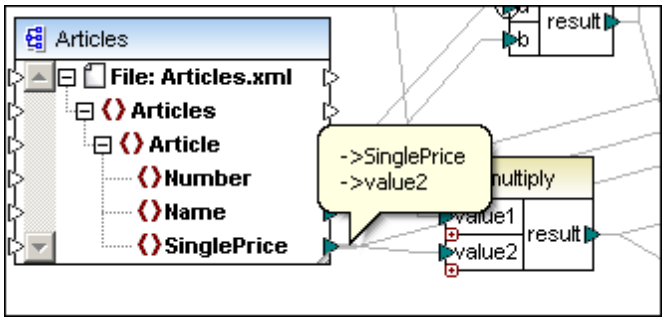


To copy a connection to a different item:

- Click the stub of the connection (the straight section closer to the target), and drag it to the destination while holding down the **Ctrl** key.

To view the item(s) at the other end of a connection:

- Point to the straight section of a connection (close to the input/output connector). A tooltip appears which displays the name(s) of the item(s) at the other end of the connection. If multiple connections have been defined from the same output, then a maximum of ten item names are displayed. In the sample below, the two target items are **SinglePrice** and **value2** of the multiply function.



To change the connection settings, do one of the following:

- On the **Connection** menu, click **Properties** (this menu item becomes enabled when you select a connection).
- Double-click the connection.
- Right-click the connection, and then click **Properties**.

See also Connection Settings.

To delete a connection, do one of the following:

- Click the connection, and then press the **Delete** key.
- Right-click the connection, and then click **Delete**.

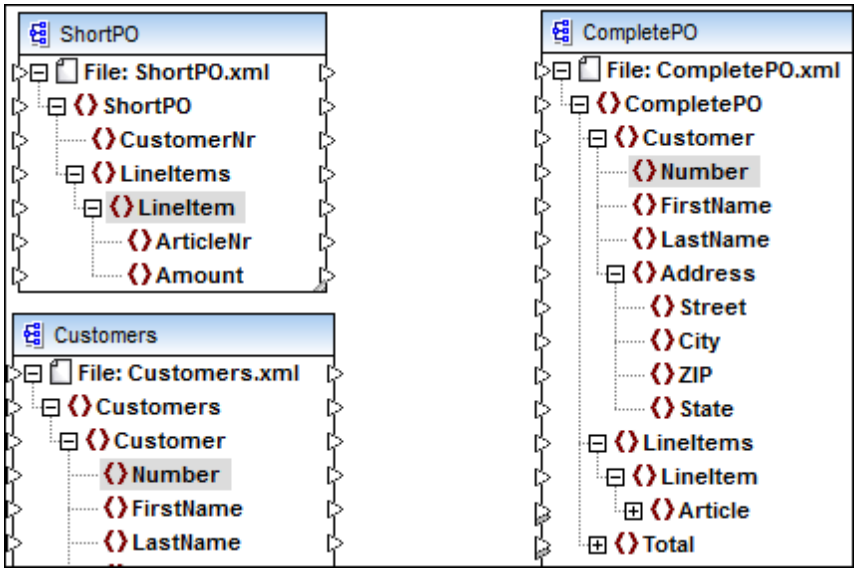
4.3.1 About Mandatory Inputs

To aid you in the mapping process, MapForce highlights in orange the mandatory inputs in target components:

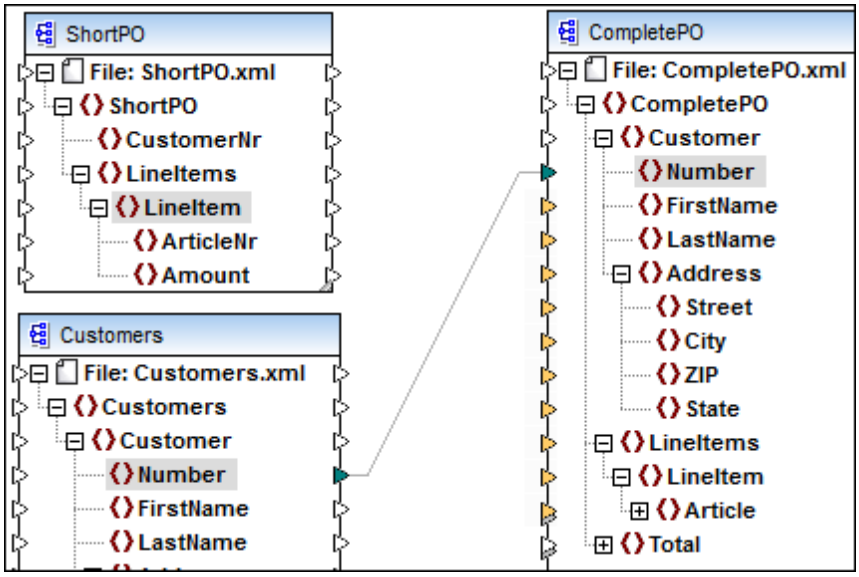
- In XML and EDI components these are items where the minOccurs parameter is equal/ greater than 1.
- In databases these are fields that have been defined as "not null"
- WSDL calls and WSDL response (all nodes)
- XBRL nodes that have been defined as mandatory
- In functions these are the specific mandatory parameters such that once one parameter has been mapped, then the other mandatory ones will be highlighted to show that a connection is needed. E.g. once one of the filter input parameters is mapped, then the other one is automatically highlighted.
- Worksheet names in MS Excel sheets

Example:
When creating a mapping like CompletePO.mfd, available in the ...\MapForceExamples folder, the

inserted XML Schema files exist as shown below.



The Number element of the Customers component is then connected to the Number element of the CompletePO component. As soon as the connection has been made, the mandatory items/ nodes of the CompletePO component are highlighted. Note that the collapsed "Article" node/icon is also highlighted.



4.3.2 Changing the Connection Display Preferences

You can selectively view the connections in the mapping window.

Show selected component connectors switches between showing:

- all mapping connectors in black, or
- those connectors relating to the currently selected component in black. Other connectors appear dimmed.

Show connectors from source to target switches between showing:

- connectors that are **directly** connected to the currently selected component, or
- connectors linked to the currently selected component, originating from source and terminating at the target components.

4.3.3 Annotating Connections

Individual connections can be labeled allowing you to comment your mapping in great detail. This option is available for **all connection types**.

To annotate to a connection:

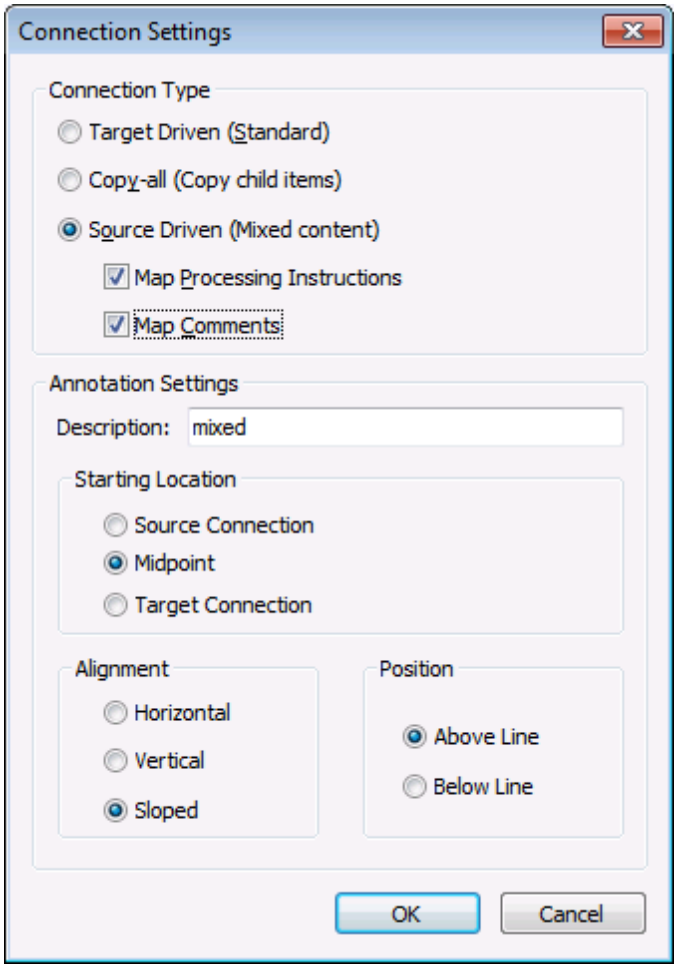
1. Right-click the connection, and select **Properties** from the context menu.
2. Enter the name of the currently selected connection in the **Description** field. This enables all the options in the Annotation Settings group.
2. Use the remaining groups to define the **starting location**, **alignment** and **position** of the label.
3. Activate the **Show annotations** icon in the View Options toolbar to see the annotation text.

Note: If the **Show annotations** icon is inactive, you can still see the annotation text if you place the mouse cursor over the connection. The annotation text will appear in a callout if the **Show tips** toolbar button is active in the View Options toolbar.

4.3.4 Connection Settings

Right-clicking a connection and selecting **Properties** from the context menu, or double-clicking a connection, opens the Connection Settings dialog box in which you can define the settings of the current connection. Note that unavailable options are disabled.

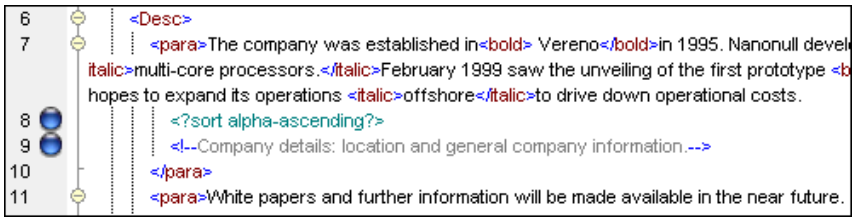
© 2015 Altova GmbH



Connection Settings dialog box

For items of **complexType**, you can choose one of the following connection types for mapping (note that these settings also apply to **complexType** items which do not have any text nodes):

Target Driven (Standard)	Changes the connection type to "Target-driven" (see Target-driven / Standard mapping).
Copy-all (Copy child items)	Changes the connection type to "Copy-all" and automatically connects all identical items in the source and target components (see Copy-all connections).
Source Driven (mixed content)	Changes the connection type to "Source-driven", and enables the selection of additional elements to be mapped. The additional elements must be child items of the mapped item in the XML source file, to qualify for mapping. Activating the **Map Processing Instructions** or **Map Comments** check boxes enables you to include these data groups in the output file.

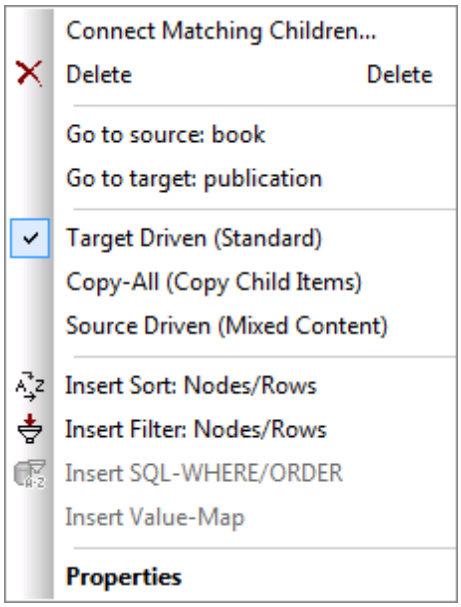


Note: CDATA sections are treated as text.

The Annotation Settings group enables you to annotate the connection (see Annotating Connections).

4.3.5 Connection Context Menu

When you right-click a connection, the following context commands are available.



Connect matching children	Opens the "Connect Matching Children" dialog box (see Connecting Matching Children). This command is enabled when the connection is eligible to have matching children.
Delete	Deletes the selected connection.
Go to source: <item name>	Selects the source connector of the current connection.
Go to target: <item name>	Selects the target connector of the current connection.
Target Driven (Standard)	Changes the connection type to "Target-driven" (see Target-driven connections).

 © 2015 Altova GmbH

Copy-All (Copy Child Items)	Changes the connection type to "Copy-all" and automatically connects all identical items in the source and target components (see Copy-all connections). This command is enabled (and meaningful) when both the source item and the target item have children items.
Source Driven (Mixed Content)	Changes the connection type to "Source-driven" (see Source-driven connections). This command is enabled (and meaningful) when both the source item and the target item have children items.
Insert Sort: Nodes/Rows	Adds a Sort component between the source and the target item (see Sorting Data).
Insert Filter: Nodes/Rows	Adds a Filter component between the source and the target item (see Filtering Data).
Insert Value-Map	Adds a Value-Map component between the source and the target item (see Using Value-Maps).
Properties	Opens the Connections Settings dialog box (see Connection Settings).

4.3.6 Connecting Matching Children

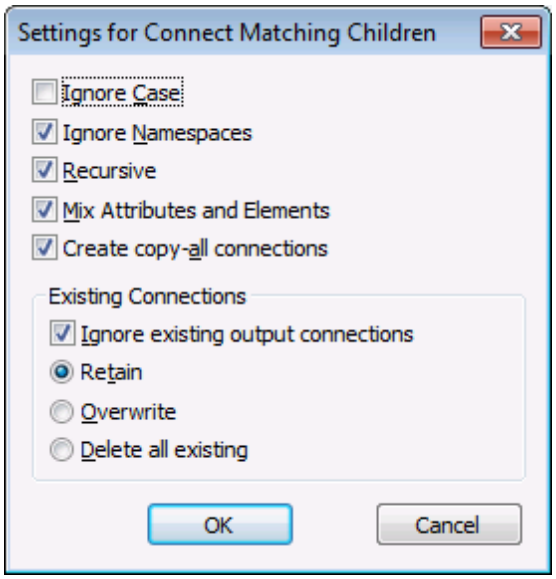
You can create multiple connections between items of the **same name** in both the source and target components. Note that a "Copy-all" connection (see Copy-all connections) is created by default.

To toggle the "Auto Connect Matching Children" option on or off, do one of the following:

- Click the **Auto Connect Matching Children** ( ) toolbar button.
- On the **Connection** menu, click **Auto Connect Matching Children**.

To change the settings for "Connect Matching Children":

1. Connect two (parent) items that share identically named **child items** in both components.
2. Right click the connection and select the **Connect matching child elements** option.



3. Select the required options (see the table below), and click OK. Connections are created for all the child items that have identical names and adhere to the settings defined in the dialog box.

Note: The settings you define here are applied when connecting two items if the **Toggle auto connect of children** ( ) toolbar button is active.

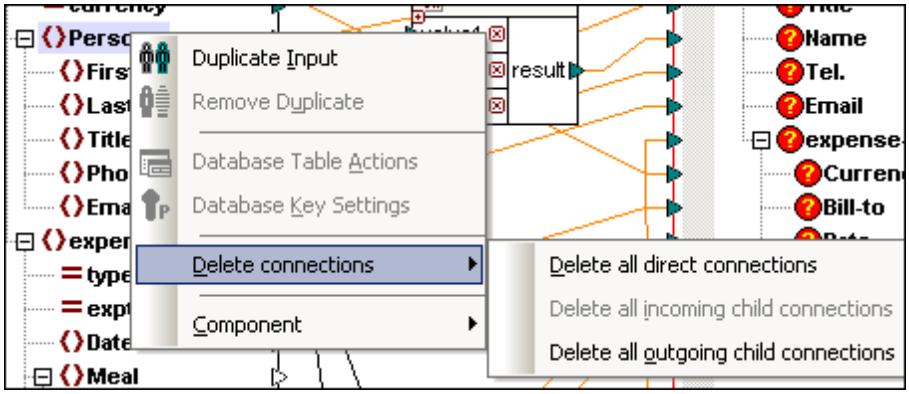
Ignore Case	Ignores the case of the child item names.
Ignore Namespaces	Ignores the namespaces of the child items.
Recursive	Creates new connections between any matching items recursively. That is, a connection is created no matter how deep the items are nested in the hierarchy, as long as they have the same name.
Mix Attributes and Elements	When enabled, allows connections to be created between attributes and elements which have the same name. For example, a connection is created if two "Name" items exist, even though one is an element, and the other is an attribute.
Create copy-all connections	This setting is active by default. It creates (if possible) a connection of type "Copy-all" between source and target items.
Ignore existing output connections	Creates additional connections for any matching items, even if they already have outgoing connections.
Retain	Retains existing connections.
Overwrite	Recreates connections according to the settings defined. Existing connections are discarded.

 © 2015 Altova GmbH

Delete all existing	Deletes all existing connections, before creating new ones.

Deleting connections

Connections that have been created using the Connect Matching Children dialog, or during the mapping process, can be removed as a group.



To delete connections:

1. Right-click the item name in the component, not the connection itself ("Person" in this example).
2. Select **Delete Connections | Delete all ... connections**.

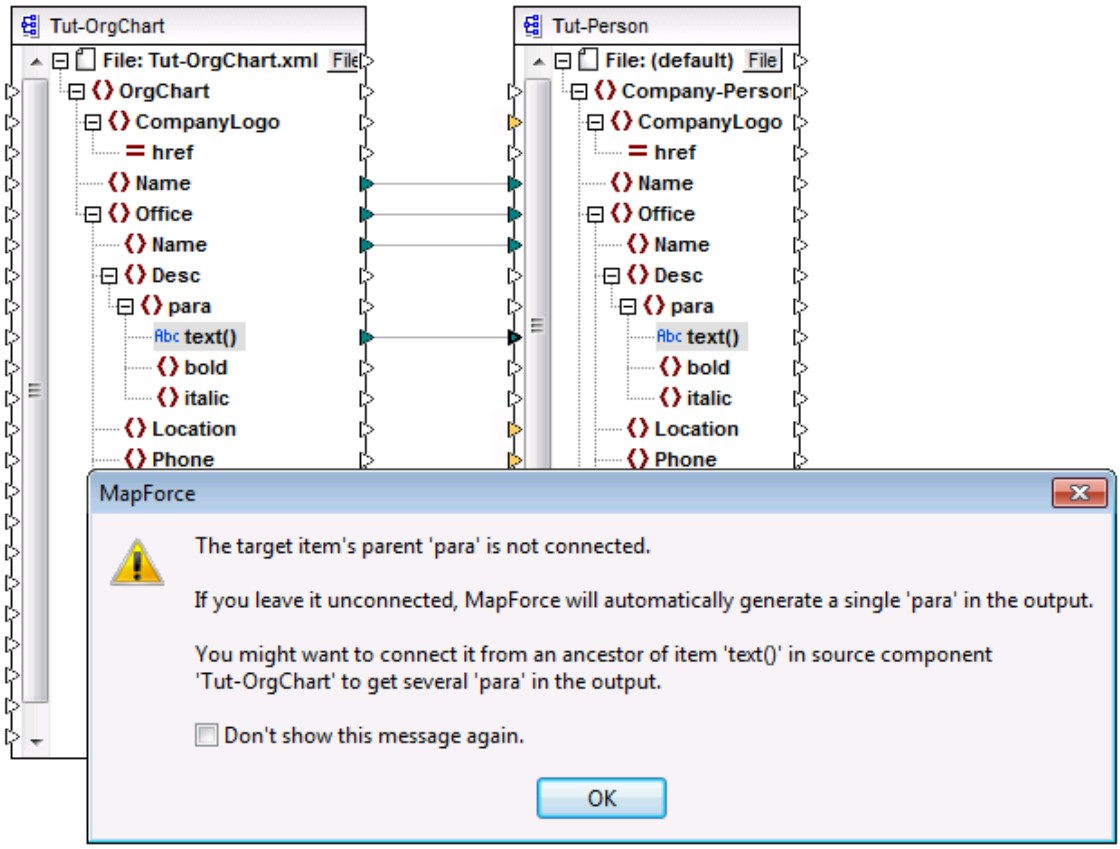
Delete all direct connections	Deletes all connections directly mapped to, or from, the current component to any other source or target components.
Delete all incoming child connections	Only active if you have right clicked an item in a target component. Deletes all incoming child connections.
Delete all outgoing child connections	Only active if you have right clicked an item in a source component. Deletes all outgoing child connections.

4.3.7 Notifications on Missing Parent Connections

When you create connections between source and target items manually, MapForce automatically analyzes the possible mapping outcomes. If you are mapping two child items, a notification message can appear suggesting that you also connect the parent of the source item with the parent in the target item.

This notification message helps you prevent situations where a single child item appears in the Output window when you preview the mapping. This will generally be the case if the source node supplies a sequence instead of a single value.

To understand how this works, open the sample mapping **Tut-OrgChart.mfd** available in the **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial** folder. If you connect the source `text()` item to the target `text()` item, a notification message appears, stating that the parent item "para" is not connected and will only be generated once in the output.



Tut-OrgChart.mfd (MapForce Basic Edition)

To generate multiple `para` items in the target, connect the source and target `para` items to each other.

To disable such notifications, do the following:

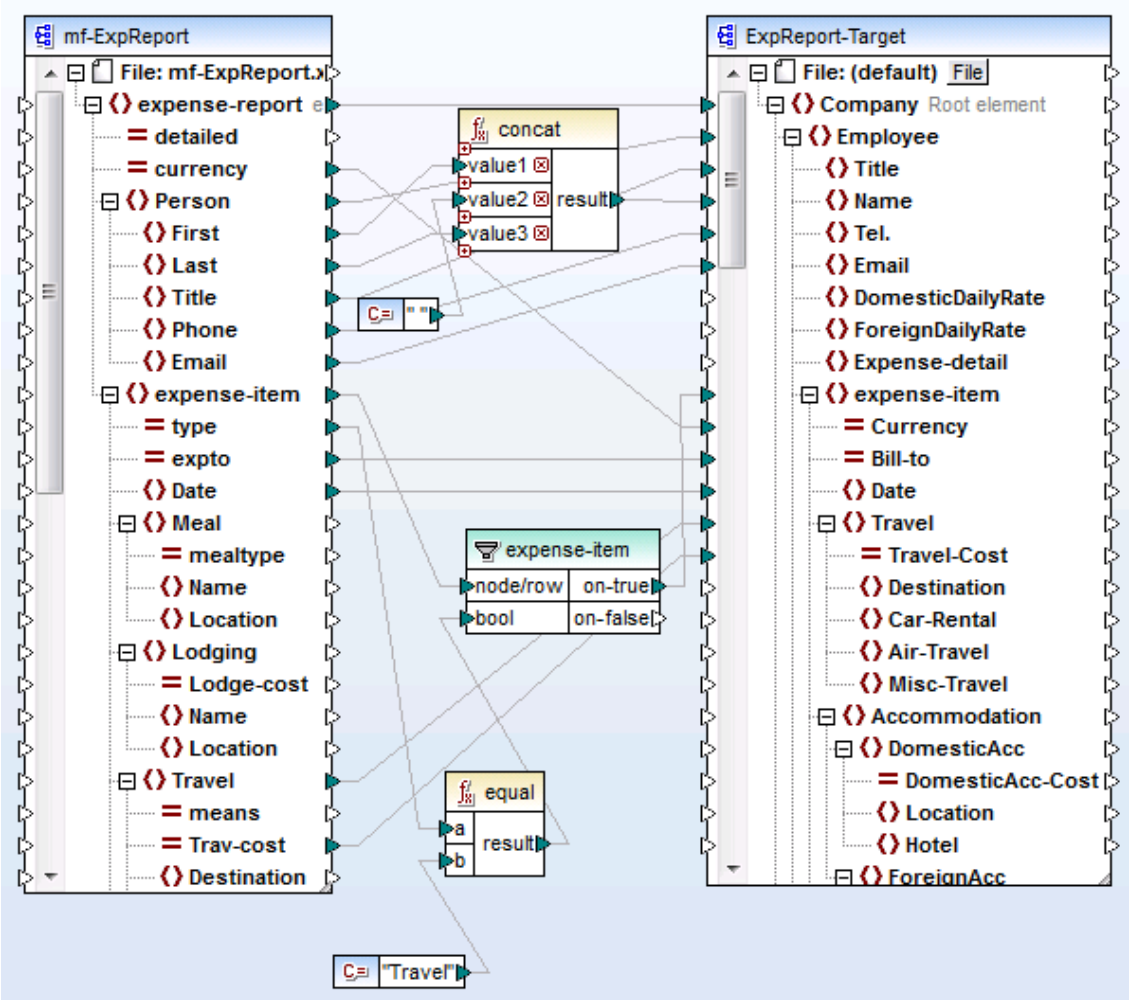
1. On the **Tools** menu, click **Options**.
2. Click the **Messages** group.
3. Click to clear the **When creating a connection, suggest connecting ancestor items** check box.

4.3.8 Moving Connections and Child Connections

When you move a connection to a different component, MapForce automatically matches identical child connections and will prompt you whether it should move them to the new location as well. A common use of this feature is if you have an existing mapping and then change the root element of the target schema. Normally, when this happens, you would need to remap all descending connections manually. This feature helps you prevent such situations.

This example uses the **Tut-ExpReport.mfd** file available in the **<Documents>\Altova \MapForce2016\MapForceExamples\Tutorial** folder.

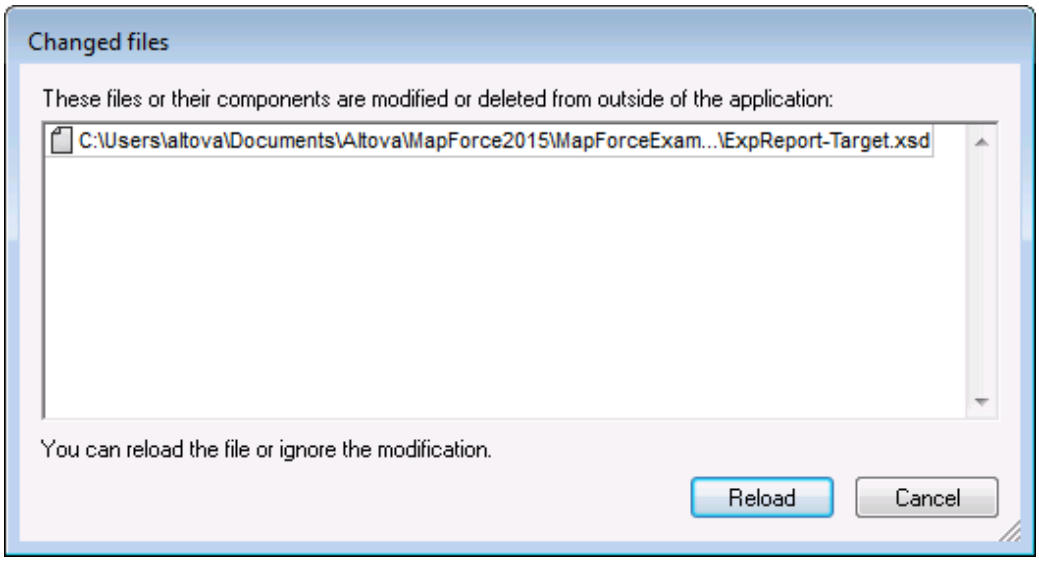
 © 2015 Altova GmbH



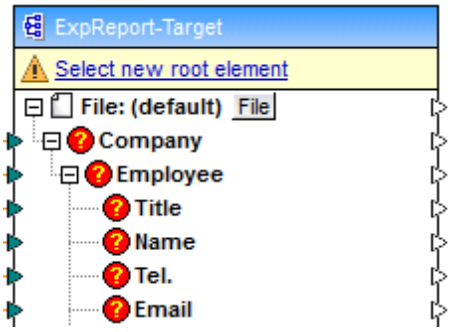
Tut-ExpReport.mfd (MapForce Basic Edition)

To understand how it works, do the following:

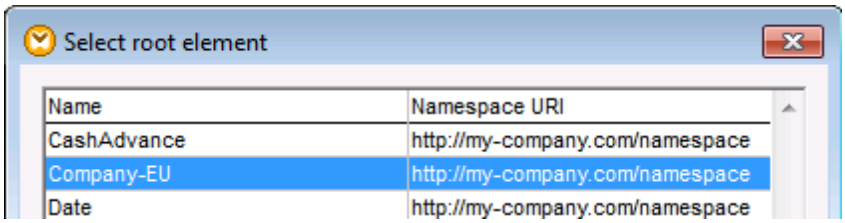
1. Open the **Tut-ExpReport.mfd** sample mapping.
2. Edit the **ExpReport-Target.xsd** schema outside MapForce so as to change the `Company` root element of the target schema to `Company-EU`. You do not need to close MapForce.
3. After you have changed the `Company` root element of the target schema to `Company-EU`, a "Changed files" prompt appears in MapForce.


Changed files
These files or their components are modified or deleted from outside of the application:
C:\Users\altova\Documents\Altova\MapForce2015\MapForceExam...\ExpReport-Target.xsd
You can reload the file or ignore the modification.
Reload
Cancel

4. Click the **Reload** button to reload the updated Schema. Since the root element was deleted, the component displays multiple missing nodes.

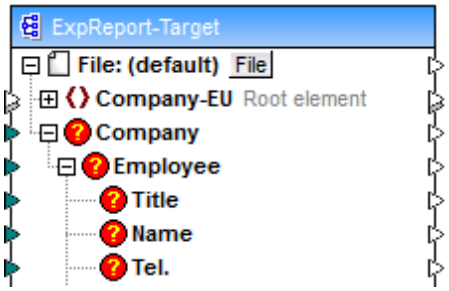


5. Click **Select new root element** at the top of the component. (You can also change the root element by right clicking the component header and selecting **Change Root Element** from the context menu.)

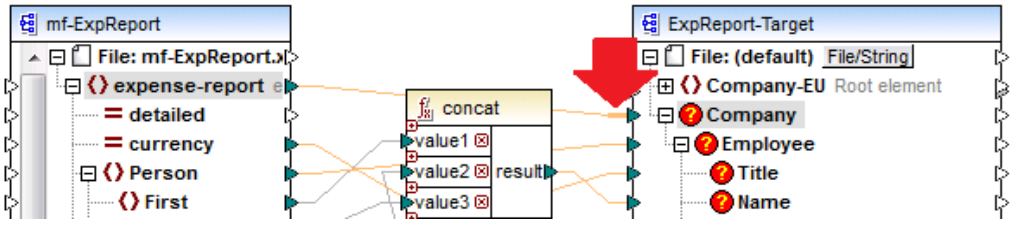


6. Select `Company-EU` as new root element and click OK to confirm. The `Company-EU` root element is now visible at the top of the component.

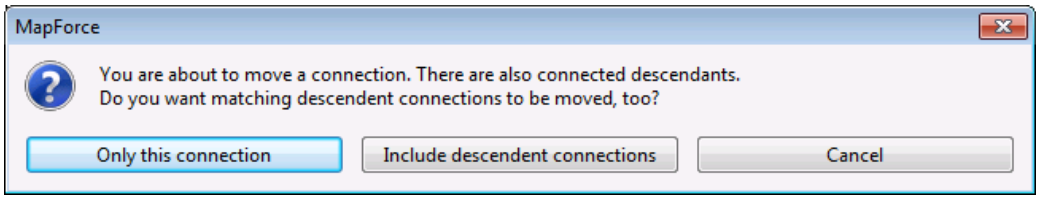
© 2015 Altova GmbH



7. Click the target stub of the connection that exists between the `expense-report` item of the source component and the `Company` item of the target component, and then drag-and-drop it on the `Company-EU` root element of the target component.



A notification dialog box appears.


MapForce
You are about to move a connection. There are also connected descendants.
Do you want matching descendent connections to be moved, too?
Only this connection
Include descendent connections
Cancel

8. Click **Include descendent connections**. This instructs MapForce to re-map the correct child items under the new root element, and the mapping becomes valid again.

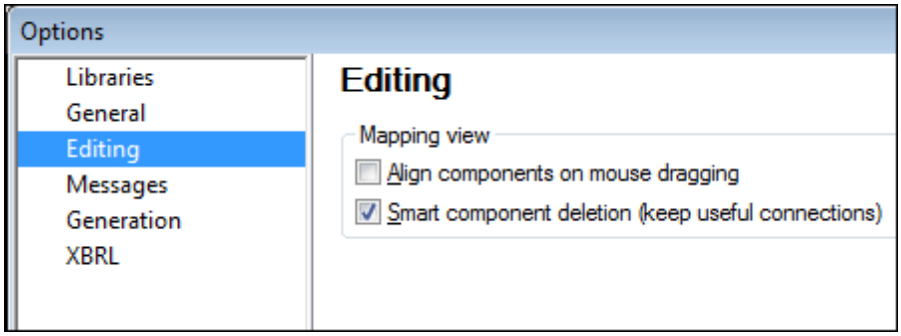
Note: If the node to which you are mapping has the same name as the source node but is in a different namespace, then the notification dialog box will contain an additional button: "Include descendants and map namespace". Clicking this button moves the child connections of the same namespace as the source parent node to the same child nodes under the different namespace node.

4.3.9 Keeping Connections After Deleting Components

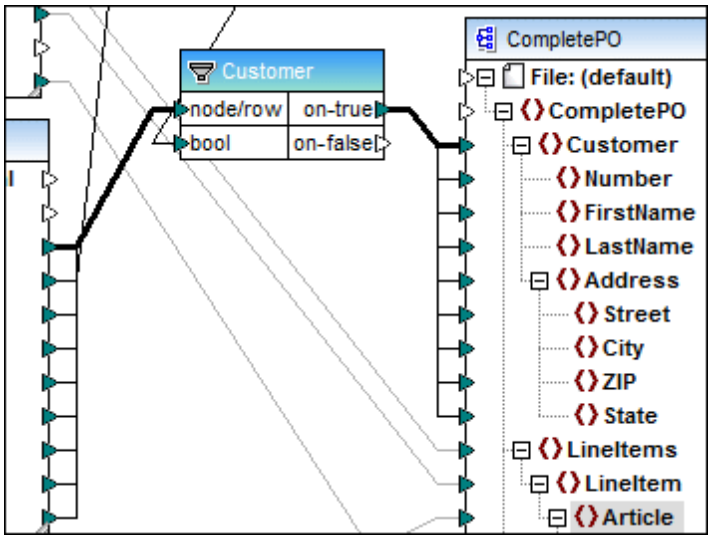
You can decide what happens when you delete a component that has multiple (child) connections to another component, e.g. a filter or sort component. This is very useful if you want to keep all the child connections and not have to restore each one individually.

You can opt to keep/restore the child connections after the component is deleted, or to delete all child connections immediately.

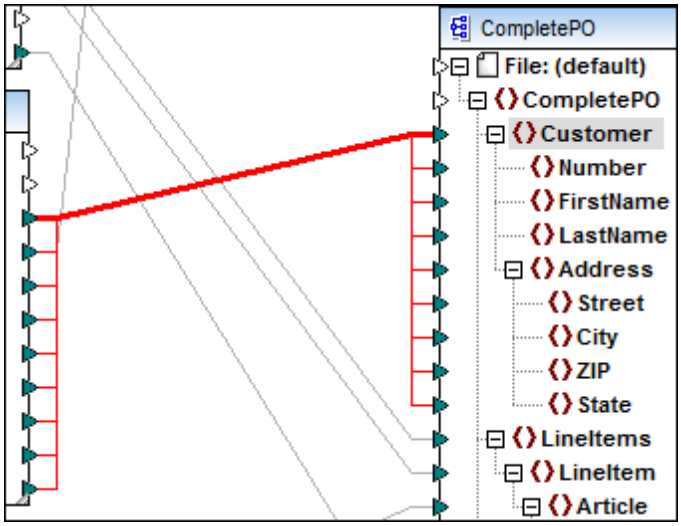
Select **Tools | Options | Editing** (tab) to see the current setting. The default setting for the check box is **inactive**, i.e. "Smart component deletion (keep useful connections)" is disabled.



E.g. using the CompletePO.mfd mapping in the ...\MapForceExamples folder, and the check box is active, the Customer filter is a **copy-all** connection with many connected child items, as shown below.



Deleting the Customer filter opens a prompt asking if you really want to delete it. If you select Yes, then the filter is deleted but all the child connectors remain.



Note that the remaining connectors are still selected (i.e. shown in red). If you want to delete them as well, hit the Del. key.

 © 2015 Altova GmbH

Clicking anywhere in the mapping area deselects the connectors.

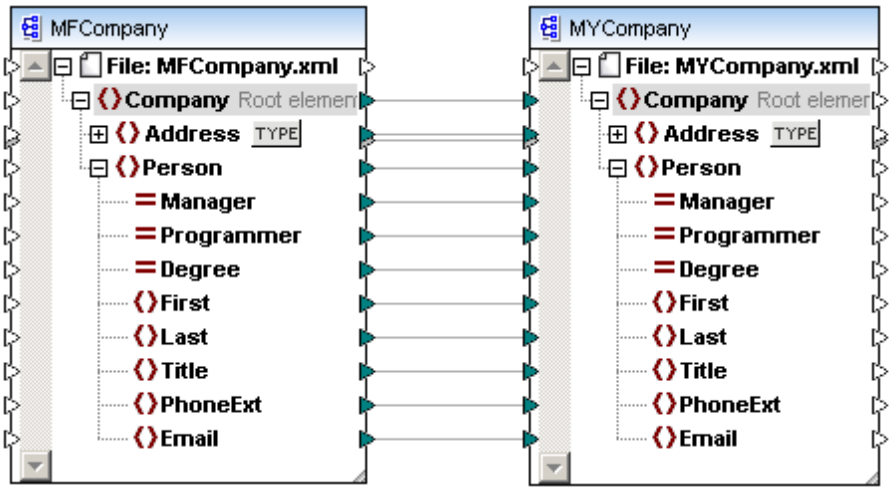
If the "Smart component deletion..." check box is **inactive**, then deleting the filter will delete all child connectors immediately.

Note: If a filter component has both "on-true" and "on-false" outputs connected, then the connectors for both outputs will be retained.

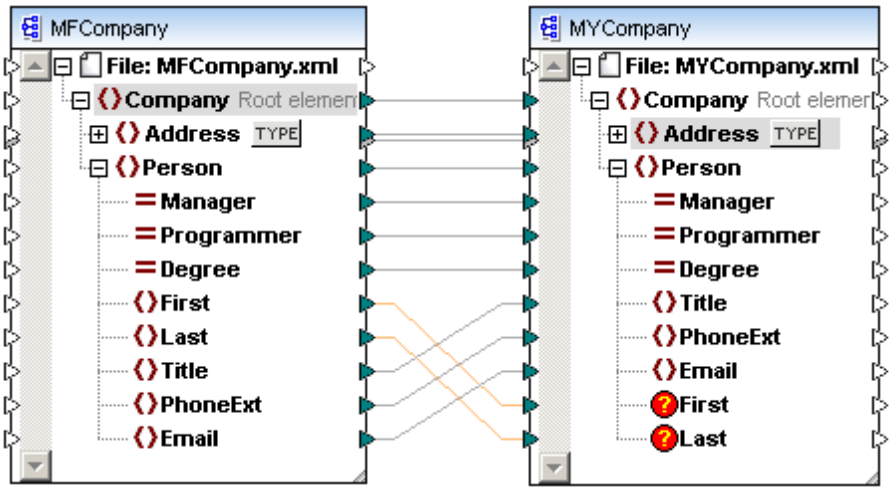
4.3.10 Dealing with Missing Items

Over time, it is likely that the structure of one of the components in a mapping may change e.g. elements or attributes are added/deleted to an XML schema. MapForce uses placeholder items to retain all the connectors, and any relevant connection data between components, when items have been deleted.

Example:
Using the **MFCompany.xsd** schema file as an example. The schema is renamed to MyCompany.xsd and a connector is created between the Company item in both schemas. This creates connectors for all child items between the components, if the Autoconnect Matching Children is active.



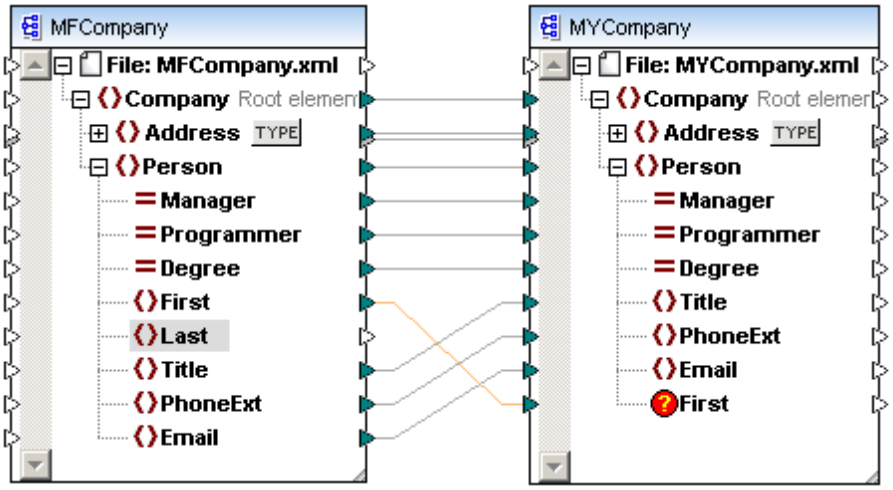
While editing MyCompany.xsd, in XMLSpy, the First and Last items in the schema are deleted. Returning to MapForce opens a Changed Files notification dialog box, prompting you to reload the schema. Clicking **Reload** updates the components in MapForce.



The deleted **items** and their **connectors** are now marked in the MyCompany component. You could now reconnect the connectors to other items if necessary, or delete the connectors.

Note that you can still preview the mapping (or generate code), but warnings will appear in the Messages window if you do so at this point. All connections to, and from, missing items are ignored during preview or code-generation.

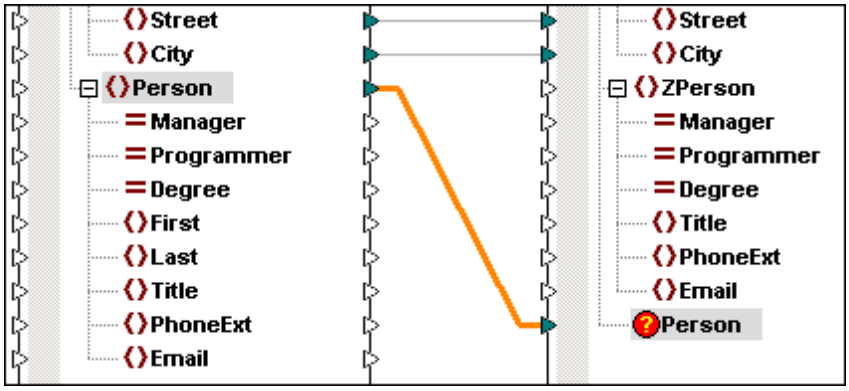
Clicking one of the highlighted connectors and deleting it, removes the "missing" item from the component, e.g. Last, in MyCompany.



Renamed items

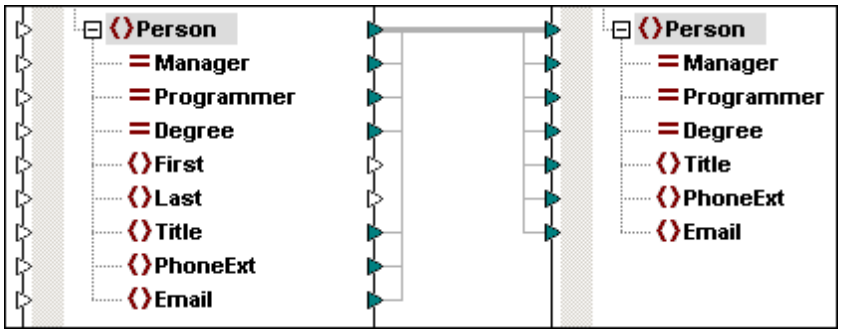
If a parent item is renamed e.g. Person to ZPerson, then the original parent item connector is retained and the child items and their connectors are deleted.

 © 2015 Altova GmbH



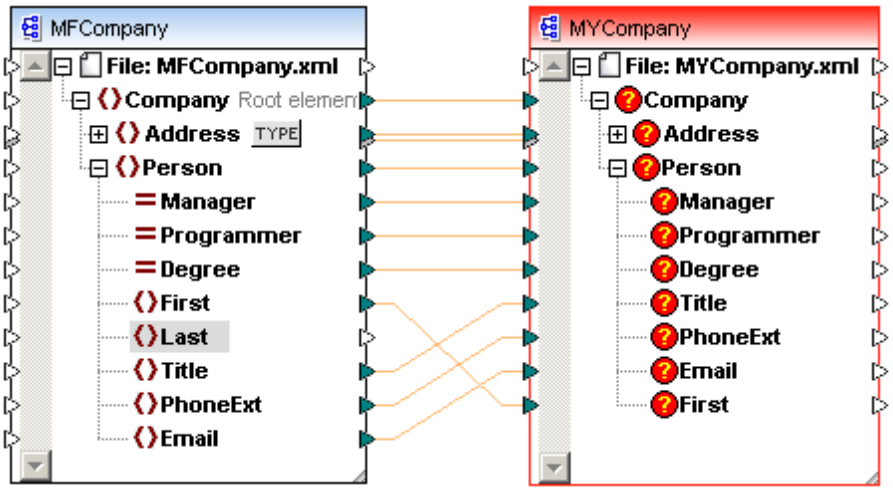
"Copy all" connectors and missing items

Copy all connections are treated in the same way as normal connections, with the only difference being that the connectors to the missing child items are not retained or displayed.

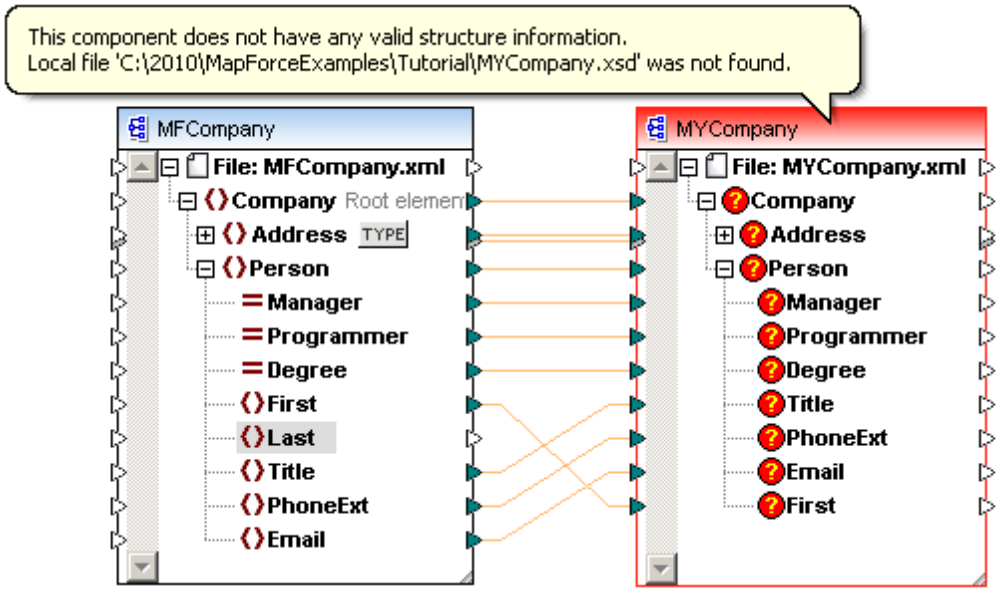


Renamed or deleted component sources

If the **data source** of a component i.e. schema has been renamed or deleted, then all items it contained are highlighted. The red frame around the component denotes that there is no valid connection to a schema and prevents preview and code generation.



Placing the mouse cursor over the highlighted component, opens a popup containing pertinent information.



Double-clicking the title bar of the highlighted component opens the Component Settings dialog box.Clicking the **Browse** button in the **Schema file** group allows you to select a different, or backed-up version of the schema. Please see "Component" in the Reference section for more information.

All valid/correct connections will be retained if you select a schema of the same structure.

© 2015 Altova GmbH

Chapter 5

Designing Mappings

5 Designing Mappings

This section describes how to design data mappings, and ways in which you can transform data on the mapping area. It also includes various considerations applicable to mapping design. Use the following roadmap for quick access to specific tasks or concepts:

I want to...	Read this topic...
Create or edit path references to miscellaneous schema, instance, and other files used by a mapping.	Using Relative and Absolute Paths
Fine-tune the data mapping for specific needs (for example, influence the sequence of items in a target component).	Connection Types
Map data from multiple sources with different schema into a single schema.	Merging data from multiple schemas
Use the output of a component as input of another component.	Chained mappings / pass-through components
Process multiple files (for example, all files within a directory) in the same mapping, either as a source or a target.	Processing Multiple Input or Output Files Dynamically
Pass an external value (such as a string parameter) to the mapping.	Supplying Parameters to the Mapping
Get a string value out of the mapping, instead of a file.	Returning String Values from a Mapping
Store some mapping data temporarily for later processing (similar to variables in a programming language).	Using Variables
Sort data in ascending or descending order.	Sorting Data
Filter data based on specific criteria.	Filtering Data
Process key-value pairs, for example, to convert months from numerical representation (01, 02, and so on) to text representation (January, February, and so on).	Using Value-Maps
Process data conditionally	Using If-Else Conditions
Learn how to avoid undesired results when designing complex mappings.	Mapping rules and strategies

Importantly, MapForce additionally includes an extensive built-in function library (see Function Library Reference) to help you with a wide array of processing tasks. When the built-in library is

 © 2015 Altova GmbH

not sufficient, you can always build your own custom functions in MapForce, or re-use external XSLT files. For further information, see Using Functions.

5.1 Using Relative and Absolute Paths

A mapping design file (*.mfd) may have references to several schema and instance files. The schema files are used by MapForce to determine the structure of the data to be mapped, and to validate it. The instance files, on the other hand, are required to read, preview, and validate the source data against the schema.

All references to files used by a mapping design are created by MapForce when you add a component to the mapping. However, you can always set or change such path references manually if required.

This section provides instructions for setting or changing the path to miscellaneous file types referenced by a mapping, and the implications of using relative versus absolute paths.

5.1.1 Using Relative Paths on a Component

The Component Settings dialog box (illustrated below for an XML component) provides the option to specify either absolute or relative paths for various files which may be referenced by the component:

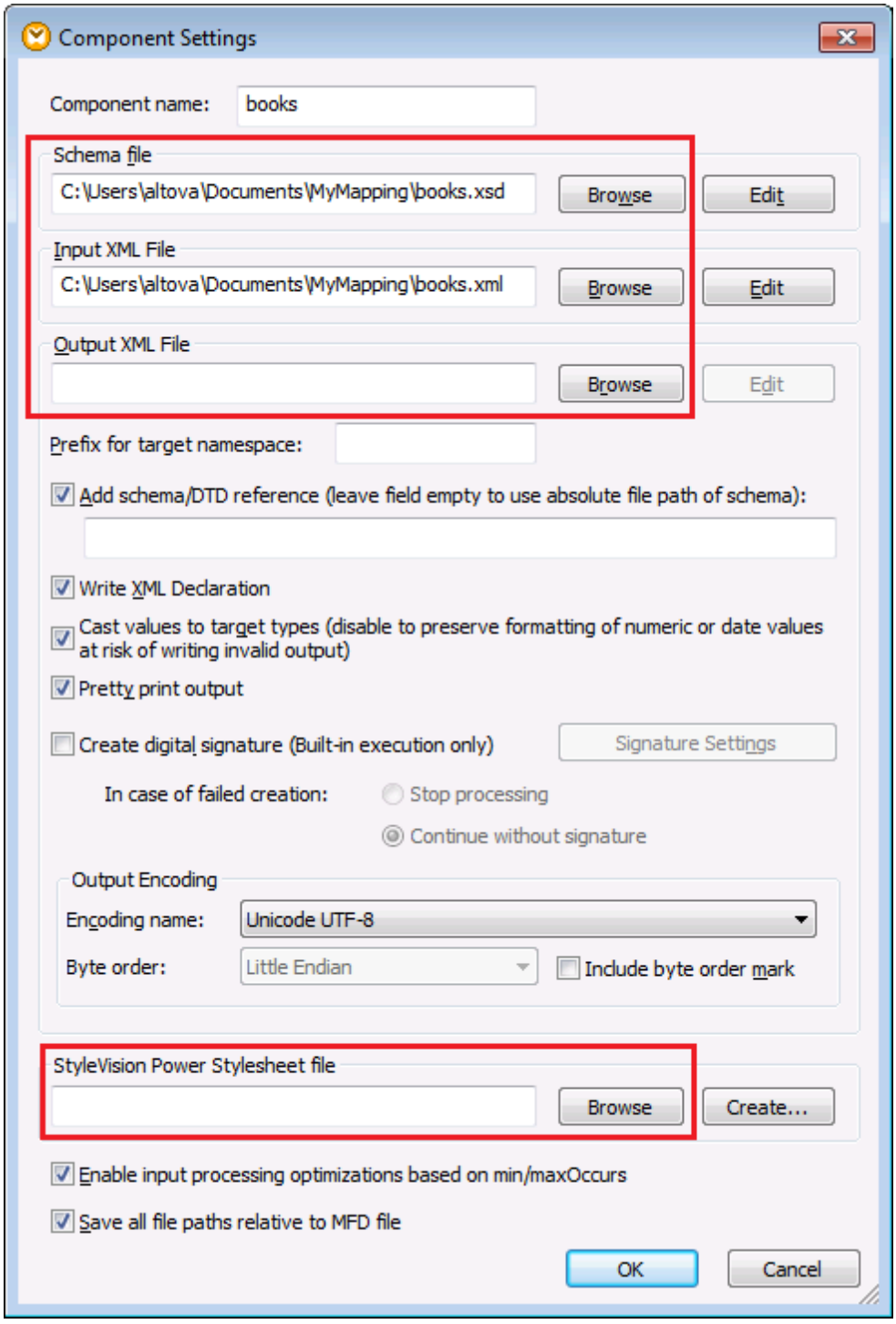
- Input files (that is, files from which MapForce reads data)
- Output files (that is, files to which MapForce writes data)
- Schema files (applicable to components which have a schema)
- Structure files (applicable to components which may have a complex structure, such as input or output parameters of user-defined functions, or variables)
- StyleVision Power Stylesheet (*.sps) files, used to format data for outputs such as PDF, HTML and Word.

You can enter relative paths directly in the relevant text boxes (shown enclosed in a red frame in the image below).

Before entering relative file paths, make sure to save the mapping file (.mfd) first. Otherwise, all relative paths are resolved against the personal application folder of Windows (Documents \Altova\MapForce2016), which may not be the intended behavior.

You can also instruct MapForce to save all above-mentioned file paths relative to the mapping .mfd file. In the sample image below, notice the option **Save all file paths relative to MFD file**. If the check box is enabled (which is the default and recommended option), the paths of any files referenced by the component will be saved relative to the path of the mapping design file (.mfd). This affects all files referenced by the component (shown enclosed in a red frame in the image).

 © 2015 Altova GmbH



Component Settings dialog box

Although the component illustrated above is an XML component, the setting **Save all file paths relative to MFD file** works in the same way for the following files:

- Structure files used by complex input or output parameters of user-defined functions and

variables of complex type
- Input or output flat files *
- Schema files referenced by database components which support XML fields *
- Input or output XBRL, FlexText, EDI, Excel 2007+, JSON files **

* MapForce Professional and Enterprise Edition
** MapForce Enterprise Edition only

Taking the component above as an example, if the .mfd file is in the same folder as the **books.xsd** and **books.xml** files, the paths will be changed as follows:

C:\Users\altova\Documents\MyMapping\books.xsd will change to **books.xsd**
C:\Users\altova\Documents\MyMapping\books.xml will change to **books.xml**

Paths that reference a non-local drive or use a URL will not be made relative.

When the check box is enabled, MapForce will also keep track of the files referenced by the component if you save the mapping to a new folder using the **Save as** menu command. Also, if all files are in the same directory as the mapping, path references will not be broken when you move the entire directory to a new location on the disk.

Using relative paths (and, therefore, enabling the **Save all file paths relative to MFD file** check box) may be important in many cases, for example:

- The location of the mapping on your operating system is likely to change in future.
- The mapping is stored in a directory which is under source control (using a version control system such as TortoiseSVN, for example).
- You intend to deploy the mapping for execution to a different machine or even to a different operating system.

If the **Save all file paths relative to MFD file** check box is disabled, saving the mapping does not modify the file paths (that is, they remain as they appear in the Component Settings dialog box).

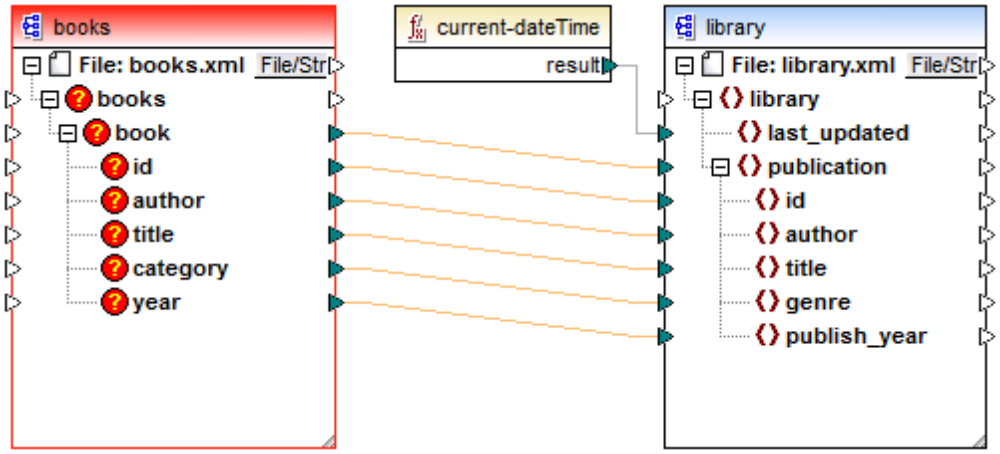
5.1.2 Fixing Broken Path References

When you add or change a file reference in a mapping, and the path cannot be resolved, MapForce displays a warning message. This way, MapForce diminishes the chance for broken path references to happen. Nevertheless, broken path references may still occur in cases such as:

- You use relative paths, and then move the mapping file to a new directory without moving the schema and instance files.
- You use absolute paths to files in the same directory as the mapping file, and then move the directory to another location.

When this happens, MapForce highlights the component in red, for example:

 © 2015 Altova GmbH



Broken path reference

The solution in this case is to double-click the component header and update any broken path references in the **Component Settings** dialog box (see also Changing the Component Settings).

5.1.3 About Paths in Generated Code

If you generate code from mappings, the generated files will no longer be run by MapForce, but by the target environment you have chosen (for example, RaptorXML Server). The implication is that, for the mapping to run successfully, any relative paths must be meaningful in the environment where the mapping runs.

Consequently, when the mapping uses relative paths to instance or schema files, consider the base path to be as follows for each target language:

Target language	Base path
XSLT/XSLT2	Path of the XSLT file.
XQuery*	Path of the XQuery file.
C++, C#, Java*	Working directory of the generated application.
BUILT-IN* (when previewing the mapping in MapForce)	Path of the mapping (.mfd) file.
BUILT-IN* (when running the mapping with MapForce Server)	The current working directory.
BUILT-IN* (when running the mapping with MapForce Server under FlowForce Server control)	The working directory of the job or the working directory of FlowForce Server.

** Languages available in MapForce Professional and Enterprise editions*

If required, you can instruct MapForce to convert all paths from relative to absolute when generating code for a mapping. This option might be useful if you run the mapping code on the same operating system, or perhaps on another operating system where any absolute path references used by the mapping can still be resolved.

To convert all paths to absolute in the generated code, enable the **Make paths absolute in generated code** check box, on the Mapping Settings dialog box (see Changing the Mapping Settings).

When you generate code and the check box is enabled, MapForce resolves any relative paths based on the directory of the mapping file (.mfd), and makes them absolute in the generated code. This setting affects the path of the following files:

- Input and output instance files for all file-based component kinds

When the check box is disabled, the file paths will be written in the generated code as they are defined in the component settings.

5.1.4 Copy-Paste and Relative Paths

When you copy a component from a mapping and paste it into another, a check is performed to ensure that relative paths of schema files can be resolved against the folder of the destination mapping. If the path cannot be resolved, you will be prompted to make the relative paths absolute by means of the folder of the source mapping. It is recommended to save the destination mapping first, otherwise relative paths are resolved against the personal application folder.

© 2015 Altova GmbH

5.2 Connection Types

When you create a mapping connection (and both the source and the target item have child items), you can optionally choose the type of the connection to be one of the following.

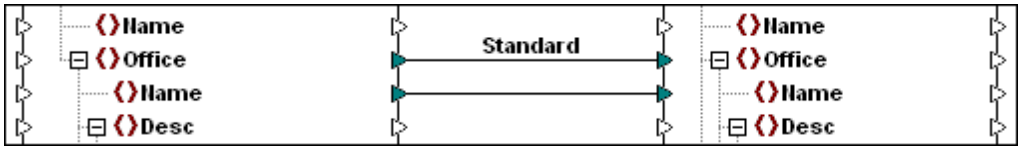
- Target Driven (Standard)
- Source Driven (Mixed Content)
- Copy-All (Copy Child Items).

The connection type determines the sequence of children items in the output generated by the mapping. This section provides information about each connection type and the scenarios when they are useful.

5.2.1 Target-driven connections

When a connection is "target-driven" (or "standard"), the sequence of child nodes in the mapping output is determined by the sequence of nodes in the target schema. This connection type is suitable for most mapping scenarios and is the default connection type used in MapForce.

On a mapping, target-driven connections are shown with a solid line.



Target-driven connections might not be suitable when you want to map XML nodes that contain mixed context (character data as well as child elements), for example:

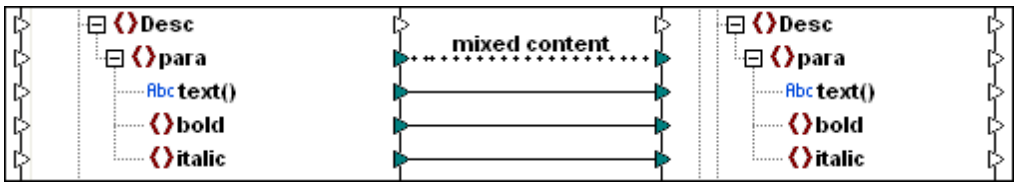
```
<p>This is our <i>best-selling</i> product.</p>
```

With mixed content, it is likely that you want to preserve the sequence of items as they appear in the source file, in which case a source-driven connection is recommended (see Source-driven connections).

5.2.2 Source-driven connections

Source-driven (Mixed Content) mapping enables you to automatically map text and child nodes in the same sequence that they appear in the XML **source** file.

- Mixed content text node content is supported/mapped.
- The sequence of child nodes is dependent on the source XML instance file.



Mixed content mappings are shown with a dotted line.

Source-driven / mixed content mapping can, of course, also be applied to XML schema **complexType** items if you wish. Child nodes will then be mapped according to their sequence in the XML source file.

Source-driven / mixed content mapping supports:

Mappings from

- As **source** components:
  - XML schema complexTypes (including mixed content, i.e. mixed=true)
- As **target** components:
  - XML schema complexTypes (including mixed content),Note: CDATA sections are treated as text.

5.2.2.1 Mapping mixed content

The files used in the following example (**Tut-OrgChart.mfd, Tut-OrgChart.mfd.xml, Tut-OrgChart.mfd.xsd, Tut-Person.xsd**) are available in the **...\MapForceExamples\Tutorial** folder.

Source XML instance

A portion of the **Tut-OrgChart.xml** file used in this section is shown below. Our area of concern is the mixed content element "para", along with its child nodes "bold" and "italic".

The `para` element also contains a Processing Instruction (`<?sort alpha-ascending?>`) as well as Comment text (`<!--Company details... -->`) which can also be mapped, as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 sp2 U (http://www.altova.com) by Mr. Nobody (Altova GmbH) -->
<OrgChart xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Tut-OrgChart.xsd">
   <CompanyLogo href="nanonull.gif"/>
   <Name>Organization Chart</Name>
   <Office>
      <Name>Nanonull, Inc.</Name>
      <Desc>
         <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull
develops nanoelectronic technologies for<italic>multi-core processors.</italic>February 1999
saw the unveiling of the first prototype <bold>Nano-grid.</bold>The company hopes to expand
its operations <italic>offshore</italic>to drive down operational costs.
            <?sort alpha-ascending?>
            <!--Company details: location and general company information.-->
         </para>
         <para>White papers and further information will be made available in the near future.
      </Desc>
```

Please note the **sequence** of the text and bold/italic nodes of Nanonull., Inc in the XML instance file, they are:

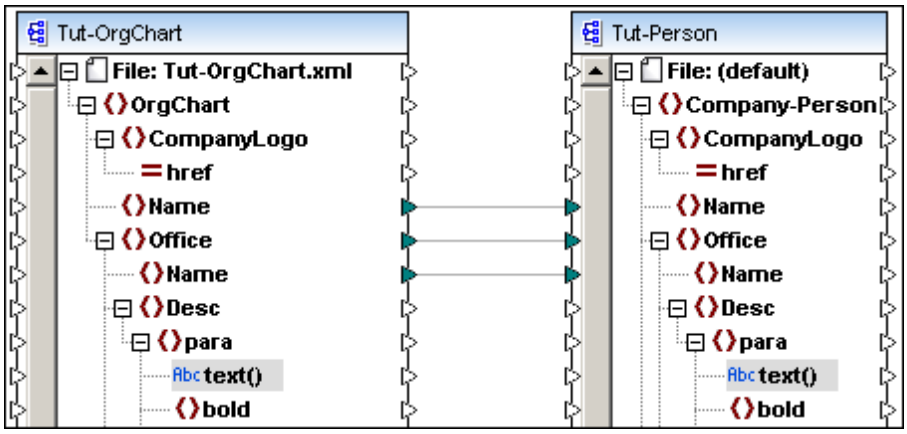
<para> The company...
<**bold**>Vereno<**/bold**>in 1995 ...
<**italic**>multi-core...<**/italic**>February 1999

 © 2015 Altova GmbH

```
        <bold>Nano-grid.</bold>The company ...
        <italic>offshore...</italic>to drive...
    </para>
```

Initial mapping

The initial state of the mapping when you open Tut-Orgchart.mfd is shown below.



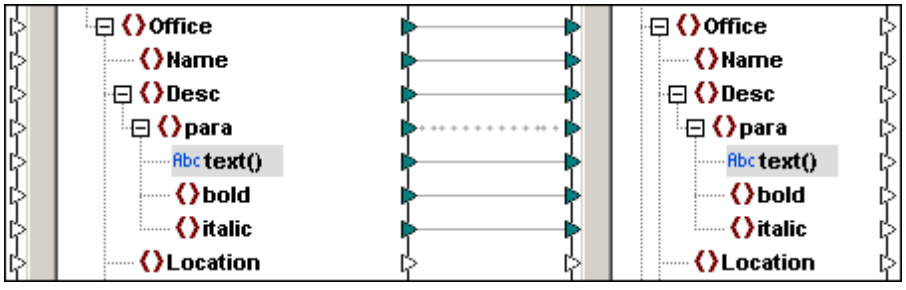
Output of above mapping

The result of the initial mapping is shown below: Organization Chart as well as the individual office names have been output.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Company-Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames
3    <Name>Organization Chart</Name>
4    <Office>
5      <Name>Nanonull, Inc.</Name>
6    </Office>
7    <Office>
8      <Name>Nanonull Europe, AG</Name>
9    </Office>
10 </Company-Person>
11
```

Mapping the para element

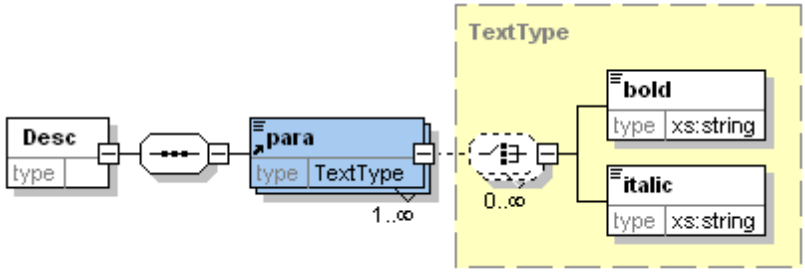
The image below shows an example of mixed content mapping. The para element is of mixed content, and the connector is shown as a **dotted** line to highlight this. The **text()** node contains the textual data and needs to be mapped for the text to appear in the target component.



To annotate (add a label to) any connection, right-click it and select **Properties** (see Annotating Connections).

The image below shows the content model of the Description element (Desc) of the **Tut-**

OrgChart.xsd schema file. This definition is identical in both the source and target schemas used in this example.



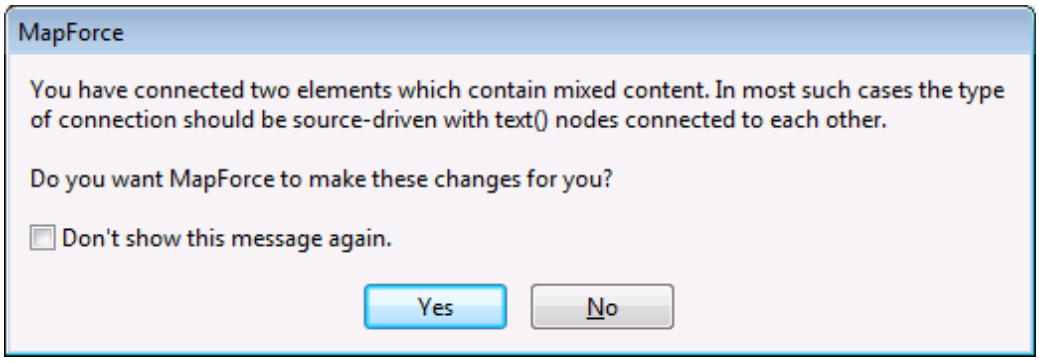
Note the following properties of the **para** element in the Content model:

- **para** is a complexType with mixed="true", of type "TextType"
- **bold** and **italic** elements are both of type "xs:string", they have not been defined as recursive in this example, i.e. neither **bold**, nor **italic** are of type "TextType"
- **bold** and **italic** elements can appear any number of times in any sequence within **para**
- any number of text nodes can appear within the **para** element, interspersed by any number of **bold** and **italic** elements.

To create mixed content connections between items:

1. Select the menu option **Connection | Auto Connect Matching Children** to activate this option, if it is not currently activated.
2. Connect the **para** item in the source schema, with the **para** item in the target schema.

   A message appears, asking if you would like MapForce to define the connectors as source driven.


MapForce
You have connected two elements which contain mixed content. In most such cases the type of connection should be source-driven with text() nodes connected to each other.
Do you want MapForce to make these changes for you?
Don't show this message again.
Yes
No

3. Click Yes to create a mixed content connection.

Please note:

Para is of mixed content, and makes the message appear at this point. The mixed-content message also appears if you only map the para items directly, without having the autoconnect option activated.

All child items of para have been connected. The connector joining the para items is displayed as a dotted line, to show that it is of type mixed content.

4. Click the Output tab to see the result of the mapping.

 © 2015 Altova GmbH

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Company-Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames
3    <Name>Organization Chart</Name>
4    <Office>
5      <Name>Nanonull, Inc.</Name>
6      <Desc>
7        <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull devel
8          </para>
9        <para>White papers and further information will be made available in the near future.
10       </para>
11     </Desc>
12   </Office>
13   <Office>
14     <Name>Nanonull Europe, AG</Name>
15     <Desc>
16       <para>In May 2000, Nanonull<italic>Europe</italic> was set up in Vienna. The team co
17     </Desc>
18   </Office>
19 </Company-Person>
20
```

5. Click the word **Wrap** icon in the Output tab icon bar, to view the complete text in the Output window.

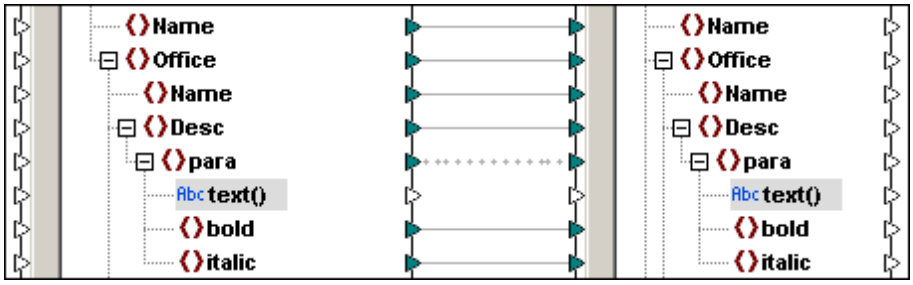
```
3    <Name>Organization Chart</Name>
4    <Office>
5      <Name>Nanonull, Inc.</Name>
6      <Desc>
7        <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull develops
     nanoelectronic technologies for<italic>multi-core processors.</italic>February 1999 saw the
     unveiling of the first prototype <bold>Nano-grid.</bold>The company hopes to expand its
     operations <italic>offshore</italic>to drive down operational costs.
8          </para>
9        <para>White papers and further information will be made available in the near future.
10       </para>
11     </Desc>
12   </Office>
13   <Office>
14     <Name>Nanonull Europe, AG</Name>
15     <Desc>
16       <para>In May 2000, Nanonull<italic>Europe</italic> was set up in Vienna. The team
     consists of<bold> five research scientists </bold>and one administrative staff.</para>
17     </Desc>
18   </Office>
19 </Company-Person>
20
```

The mixed content text of each office description has been mapped correctly; the text, as well as the bold and italic tag content, have been mapped as they appear in the XML **source** file.

6. Switch back to the Mapping view.

To remove text nodes from mixed content items:

1. Click the **text()** node connector and press Del. to delete it.



2. Click the Output tab to see the result of the mapping.

```
5     <Name>Nanonull, Inc.</Name>
6     <Desc>
7       <para>
8         <bold> Vereno</bold>
9         <italic>multi-core processors.</italic>
10        <bold>Nano-grid.</bold>
11        <italic>offshore</italic>
12      </para>
13      <para/>
14    </Desc>
15  </Office>
16  <Office>
17    <Name>Nanonull Europe, AG</Name>
18    <Desc>
19      <para>
20        <italic>Europe</italic>
21        <bold> five research scientists </bold>
22      </para>
23    </Desc>
```

Result:

- all **text** nodes of the para element have been removed.
- mapped bold and italic text content remain
- the bold and italic item **sequence** still follows that of the source XML file.

To map the Processing Instructions and Comments:

1. Right-click the mixed content connection, and select **Properties**.
2. Under **Source-Drive (Mixed content)**, select the **Map Processing Instructions** and **Map Comments** check boxes.

5.2.2.2 Mixed content example

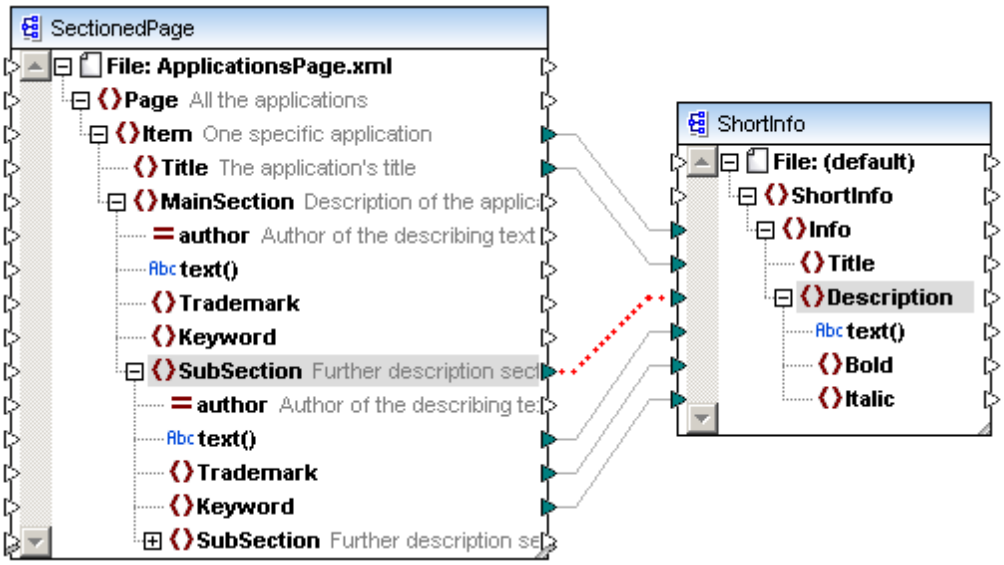
The following example is available as "**ShortApplicationInfo.mfd**" in the **...\MapForceExamples** folder.

A snippet of the XML source file for this example is shown below.

```
<Page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SectionedPage.xsd">
    <Item>
        <Title>XMLSpy</Title>
        <MainSection author="altova">
            Altova <Trademark>XMLSpy</Trademark>
            <SubSection>Altova <Trademark>XMLSpy</Trademark> 2005 Enter
is the industry standard <Keyword>XML</Keyword> development environment
editing, debugging and transforming all <Keyword>XML</Keyword> technolo
automatically generating runtime code in multiple programming languages
        </MainSection>
    </Item>
```

The mapping is shown below. Please note that:

- The "SubSection" item connector is of mixed content, and is mapped to the Description item in the target XML/schema.
- The text() nodes are mapped to each other
- Trademark text is mapped to the Bold item in the target
- Keyword text is mapped to the Italic item in the target



Mapping result

The mixed content text of each description has been mapped correctly; the text, as well as the bold and italic tag content, have been mapped as they appear in the XML source file.

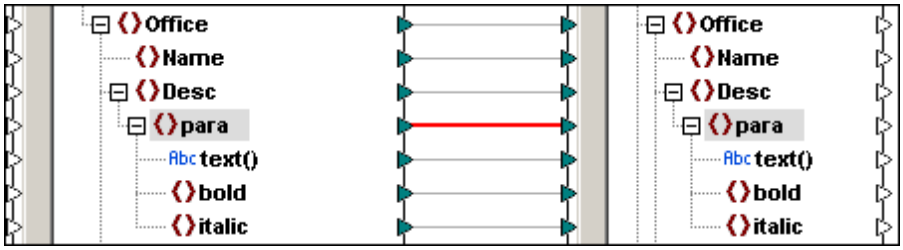
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ShortInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="
   C:/PROGRA~1/Altova/MapForce2005/MapForceExamples/ShortInfo.xsd">
3      <Info>
4          <Title>XMLSpy</Title>
5          <Description>Altova <Bold>XMLSpy</Bold> 2005 Enterprise Edition is the industry standard
   <Italic>XML</Italic> development environment for modeling, editing, debugging and transforming
   all <Italic>XML</Italic> technologies, then automatically generating runtime code in multiple
   programming languages.</Description>
6      </Info>
```

5.2.2.3 Using standard mapping on mixed content items

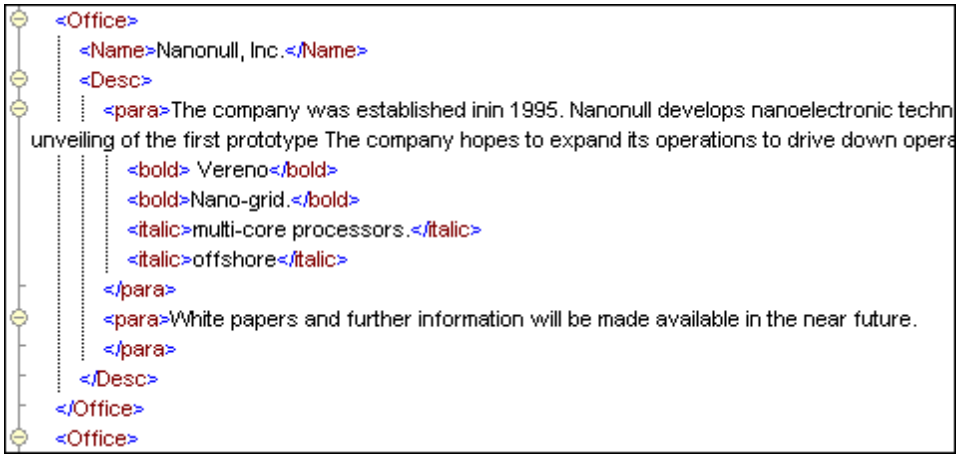
This section describes the results when defining **standard** mappings (or using standard connectors) on **mixed content** nodes. The files used in the following example (**Tut-OrgChart.mfd**) are available in the ...\MapForceExamples\Tutorial\ folder.

To create standard connections between mixed content items:

1. Create a connector between the two **para** items.
   A message appears, asking if you would like MapForce to define the connectors as source driven.
2. Click No to create a standard mapping.



3. Click the Output tab to see the result of the mapping.



Result

Mapping mixed content items using standard mapping produces the following result:

- Text() **content** is supported/mapped.
- The start/end tags of the child nodes, bold and italic, are removed from the text node.
- The child nodes appear after the mixed content node text.
- The **sequence** of child nodes depends on the sequence in the **target** XML/schema file.

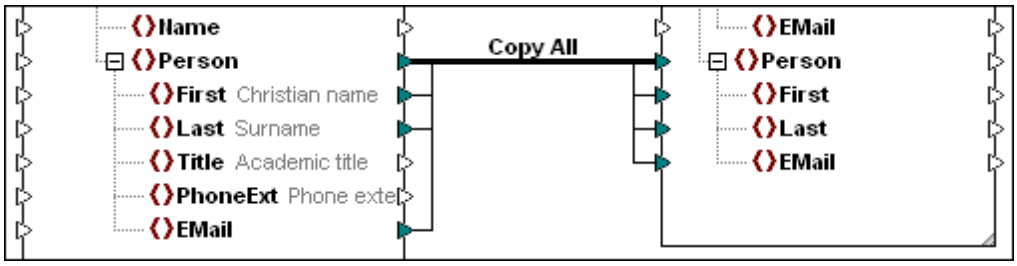
That is:
For each **para** element, map the text() node, then **all bold** items, finally **all italic** items. This results in the child item sequence shown above: bold, bold - italic, italic. The content of each item is mapped if a connector exists.

 © 2015 Altova GmbH

5.2.3 Copy-all connections

This type of connection allows you to simplify your workspace and automatically connect **all** identical items in source and target components, meaning that, depending on the source and target **type**, all source child items are **copied** to the target component, if either the source and target **types** are **identical**, or if the target type is xs:anyType.

If the source and target **types** are **not identical,** and if the target type is not xs:anyType, the source data is transferred/mapped to the respective target items of the same name and the same hierarchy level. If the names of the target items differ, then the mapping to the target item is not created.



Connections of type "Copy-All" are shown with a single bold line that connects the various identical items of source and target components.

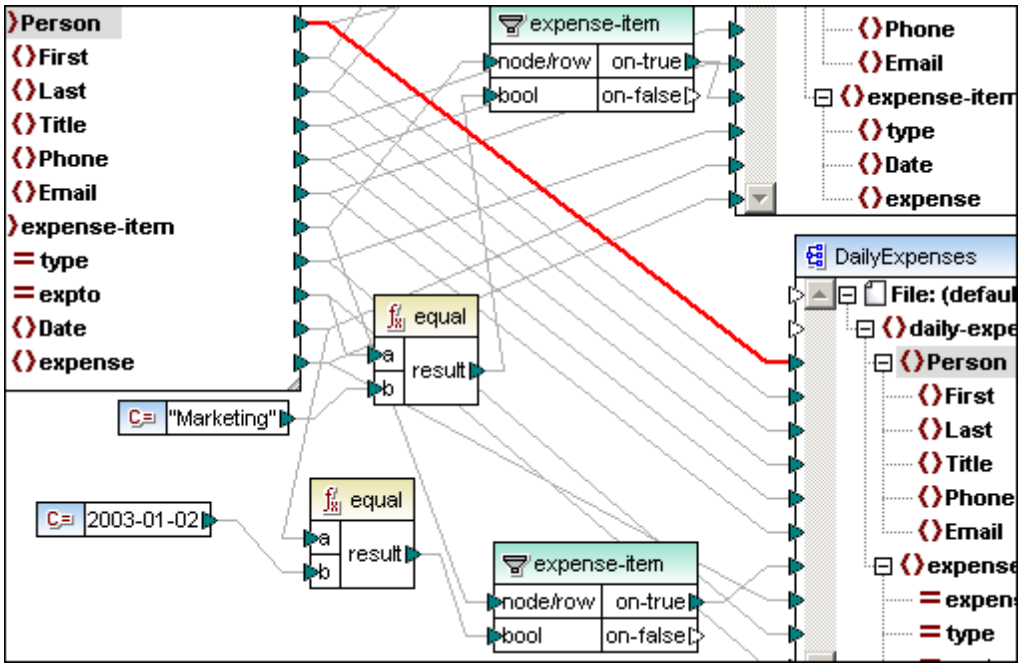
Note that only the names of the child items, but not their individual types, are compared/matched.

Currently, "Copy-All" connections are supported (i) between XML schema complex types, and (ii) between complex components (XML schema) and complex user-defined functions/components containing the same corresponding complex parameters.

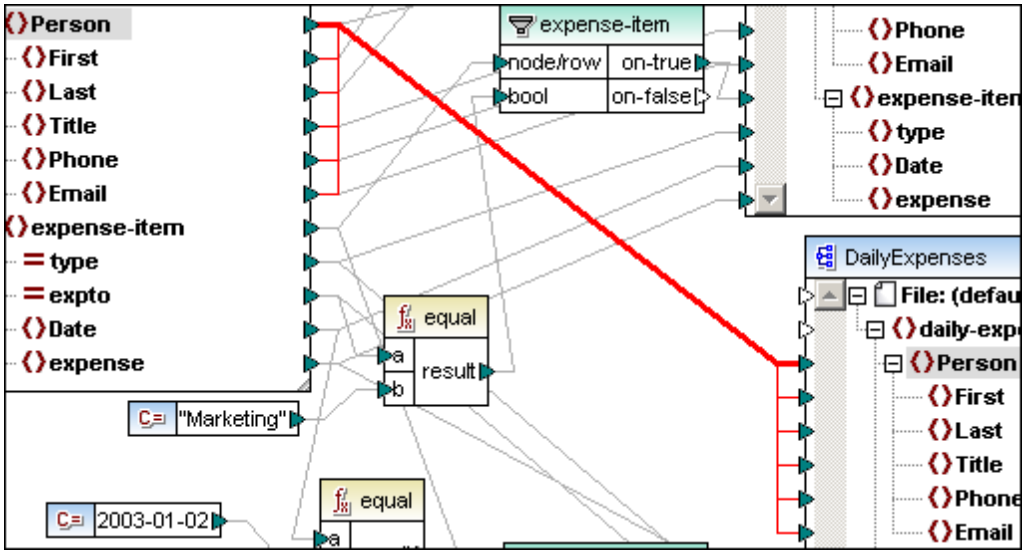
The example below uses the **MarketingAndDailyExpenses.mfd** sample available in the **...\MapForceExamples** folder.

To create a "Copy-All" connection:

1. Right-click an existing connection which has child items, and select **Copy-All (Copy Child Items)** from the context menu. A dialog box appears asking for your confirmation.



2. Click OK to create Copy-all connectors. If any child items have identical names in the source and target, connections between them are created automatically.



Please note:

- When the existing target connections are deleted, connections from other source components functions are also deleted.
- A "Copy-all" connection cannot be created between an item and the root element of a schema component.
- Individual connections cannot be deleted or reconnected from the Copy-all group, once you have used this method.

To resolve/delete "Copy-All" connections:

1. Connect any item to a child item of the copy-all connection at the target component.

 © 2015 Altova GmbH

You are notified that only one connector can exist at the target item. Click Replace to replace the connector.

2. Click the **Resolve copy-all connection** button in the next message box that opens. The copy-all connection is replaced by individual connectors to the target component.

Copy-all connections and user-defined functions

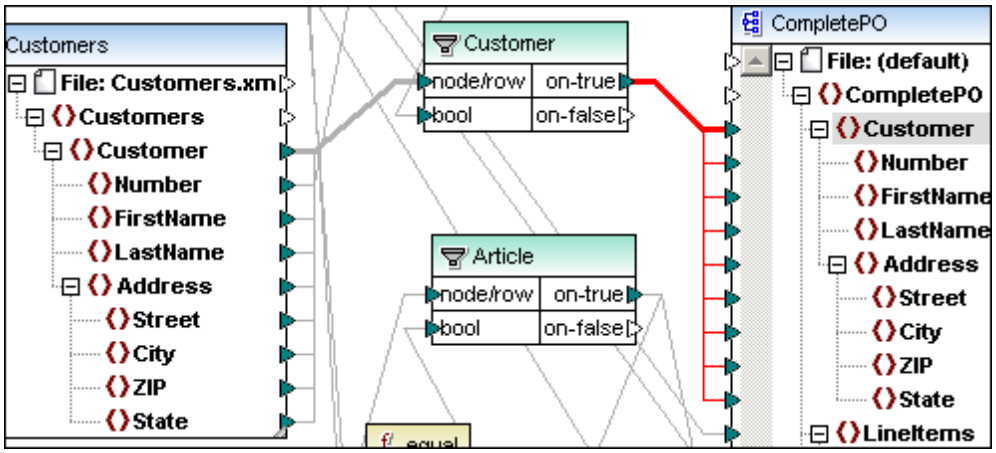
When creating Copy-all connections between a schema and a user-defined function parameter, the two components must be based on the same schema. It is not necessary that they both have the same root elements, however. Please see "Complex output components - defining" for an example.

Copy-all connections and filters

Copy-all connections can also be created through filter components if the source component:

- consists of structured data, meaning a schema component,
- receives data through a complex output parameter of a user-defined function, or Web service,
- receives data through another filter component.

Only the filtered data is passed on to the target component.



To create a copy-all connection through a filter component:

1. Create a connector from the `on-true/on-false` item to the target item, e.g. Customer.
2. Right-click the connector and select **Copy-all (Copy child items)** from the context menu.

To influence what happens when filter components are deleted, see Moving Connections and Child Connections.

5.3 Chained Mappings

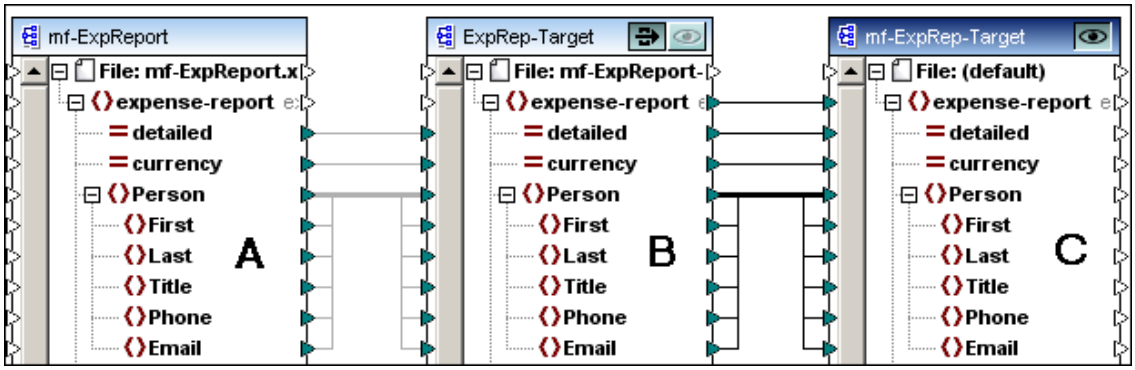
MapForce supports mappings that consist of multiple components in a mapping chain. Chained mappings are mappings where at least one component acts both as a source and a target. Such a component creates output which is later used as input for a following mapping step in the chain. Such a component is called an "intermediate" component.

Chained mappings introduce a feature called "pass-through", which allows you to create intermediate file outputs of "intermediate" components for preview, command line execution and in code generation. "Pass-through" is a preview capability allowing you to view the various stages of a chained mapping in the Output window.

If the mapping is executed from the command line, or generated code, then regardless of the entries in the Input/Output XML File fields of the "intermediate" component, the full mapping chain is executed, and the output of a previous step of a mapping chain is forwarded as input to the following mapping step.

Note: The "pass-through" feature is available only for file-based components (for example, XML, CSV, and text). Database components can be intermediate, but the pass-through button is not shown. The intermediate component is always regenerated from scratch when previewing or generating code. This would not be feasible with a database as it would have to be deleted prior to each regeneration.

The screenshot below shows three components A, B, and C, where C is the target component. Component B (ExpRep-Target) is the "intermediate" component, as it has both input and output connections.



Note that when executing a chained mapping using the command line, or executing the generated code, the mapping executes all steps in the correct order and generates the necessary output files.

Preview button

Both the component B and the component C have preview buttons. This allows you to preview the intermediate mapping result of B, as well as the final result of the chained mapping of component C in the Built-in execution engine. Click the preview button of the respective component, then click Output to see the mapping result.

"Intermediate" components with the pass-through button active cannot be previewed, since the preview button is automatically disabled. To see the output of such a component, click the "pass-

 © 2015 Altova GmbH

through" button to deactivate it, and then click the preview button of the intermediate component.

Pass-through button

The intermediate component B has an extra button in the component title bar called "pass-through".

If the pass-through button is **active** , MapForce maps all data into the preview window in one go; from component A to component B, then on to component C. Two result files will be created:

- the result of mapping component A to intermediate component B
- the result of the mapping from the intermediate component B, to target component C.

If the pass-through button is **inactive** , MapForce will execute only parts of the full mapping chain. Data is generated depending on which Preview buttons are active on the components B or C:

- If the Preview button of component B is active, then the result of mapping component A to component B is generated. The mapping chain actually stops at component B. Component C is not involved in the preview at all.
- If the Preview button of component C is active, then the result of mapping intermediate component B to the component C is generated. When pass-through is inactive, automatic chaining has been interrupted for component B. Only the right part of the mapping chain is executed. Component A is not used.

If the mapping is executed from the command line, or generated code, then, regardless of the settings of the pass-through button of component B, as well as the currently selected preview component, the output of all components is generated.

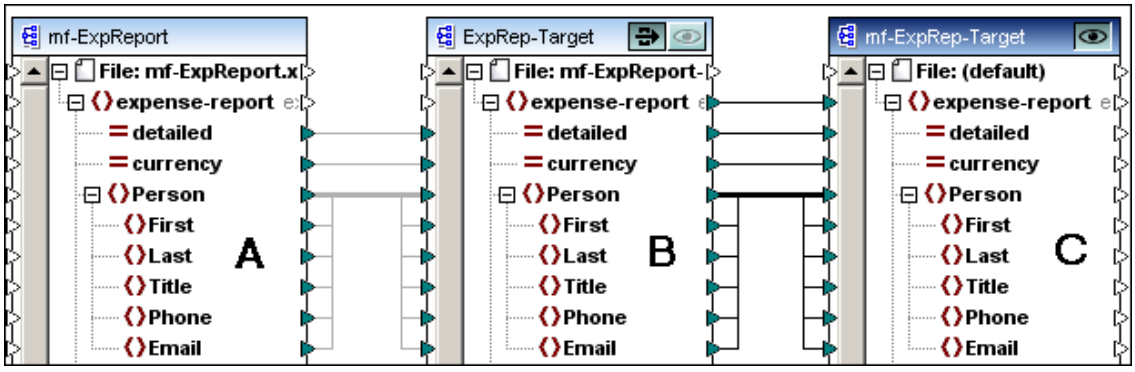
In our sample, two result files will be generated. This is the case because MapForce automatically analyzes the dependency of all components and generates all outputs of intermediate and final target components in the correct order.

Since the "pass-through" setting is currently inactive, it is vital that the intermediate component B has identical file names in the "Input XML file" and "Output XML file" fields.

Please see the following sections for more on this example, and how the source data is transferred differently when the pass-through button is active or inactive. Please see Chained mapping example for a more plausible example.

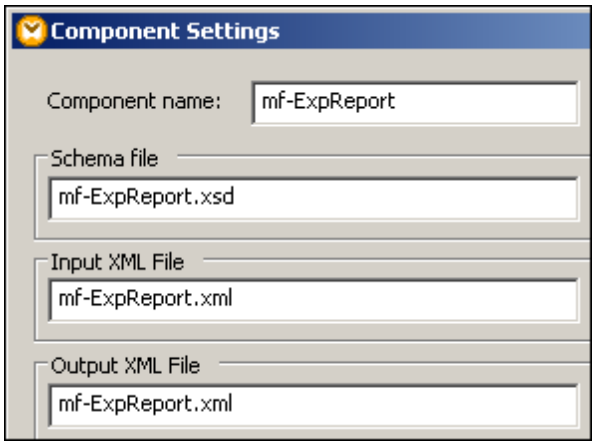
5.3.1 Chained mappings - Pass-through active

The files used in the following example (**Tut-ExpReport-chain.mfd**) are available in the **...\MapForceExamples\Tutorial** folder.



The Tut-ExpReport-chain.mfd example (*screenshot above*) is set up as follows:

- **Component A** supplies all the mapping data, using a sample XML file. The XML file (mf-ExpReport.xml) appears in the *Input XML File* field of the **Component Settings** dialog box. The Output XML File, of the same name, is automatically inserted when you define an Input XML file.

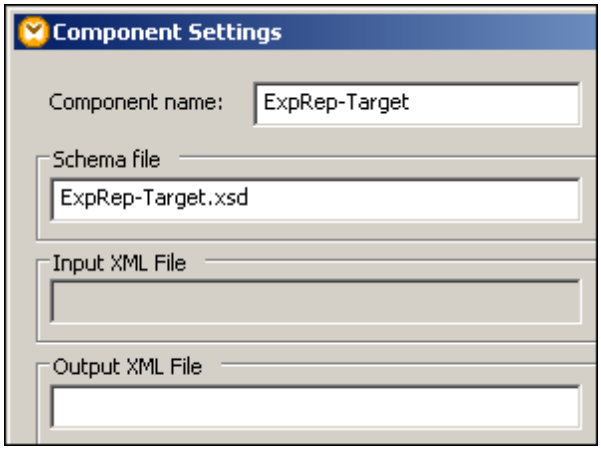


- Intermediate **component B** "pass-through" active:
  When pass-through is active, the *Input XML File* field of the intermediate component, is automatically deactivated. A file name need not exist for the mapping to execute, as intermediate data is stored in temp files.

  If no Output XML File is defined, a default file name will be automatically used. If an Output XML File entry exists, then it is used for the file name of the intermediate output file.

  Note that it is also possible for intermediate components to have dynamic file names i.e. connectors to the "File:" item of a component (or even file name wildcards). See Processing Multiple Input or Output Files Dynamically.

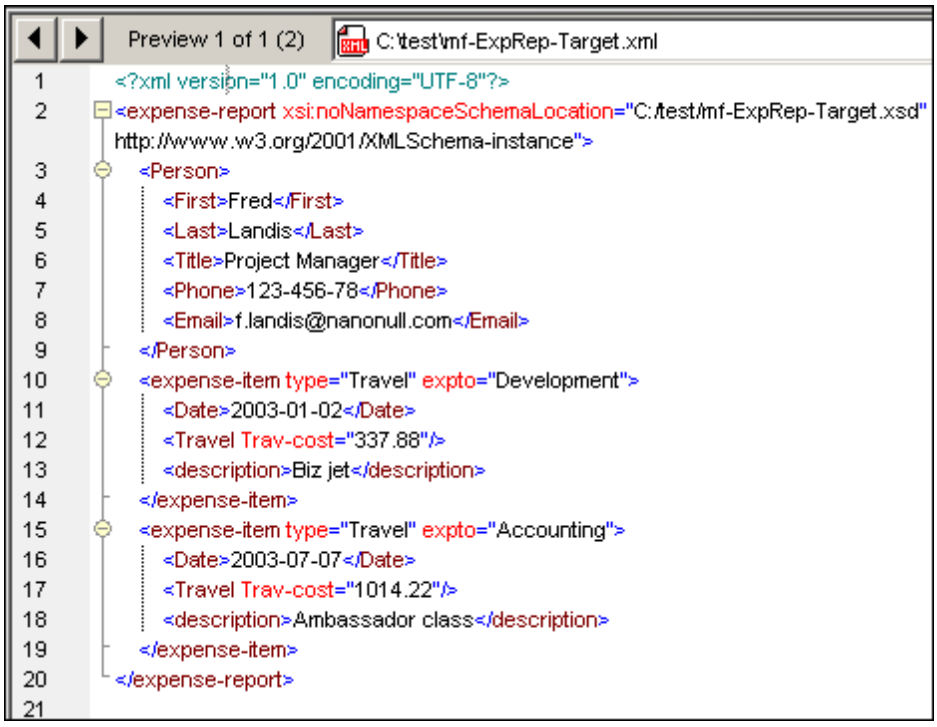
 © 2015 Altova GmbH



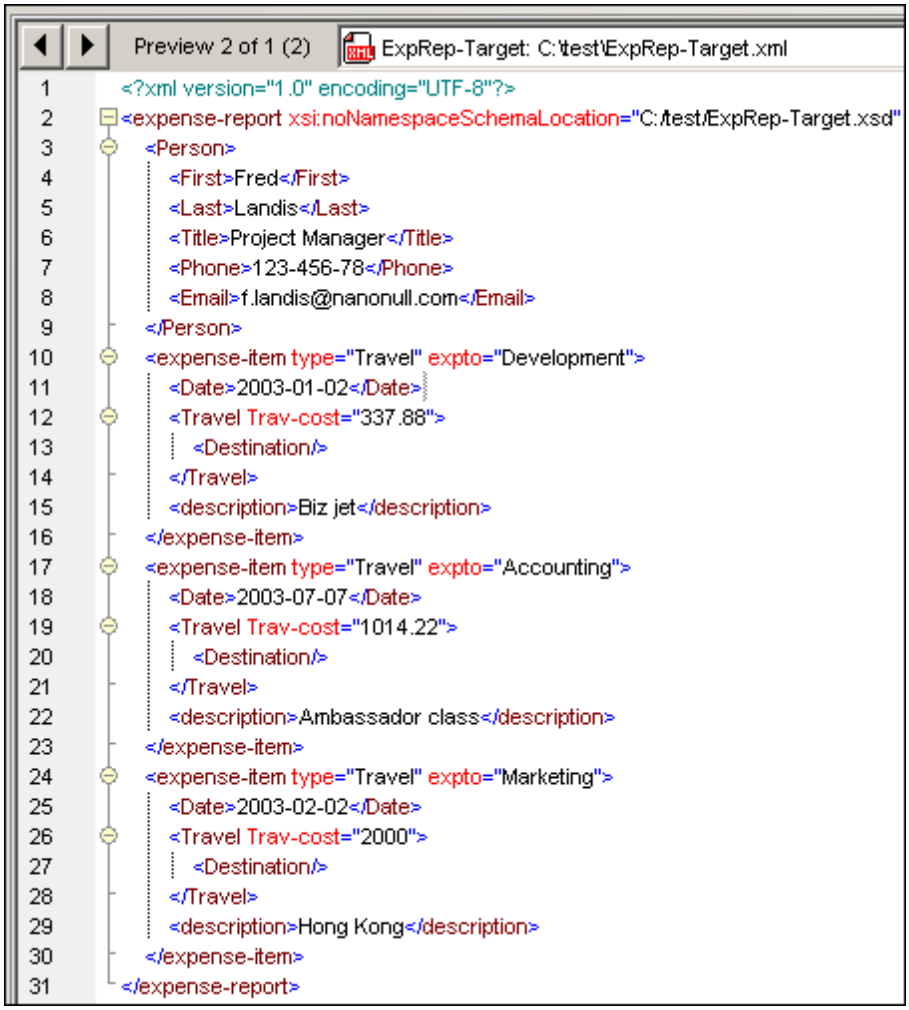
- Final **component C** does **not** have an Output XML File assigned to it. The preview button of component C is active.

Click the Output button to preview the results in the Built-in execution engine.

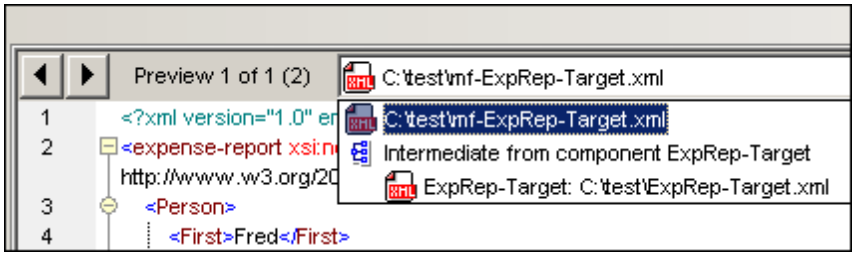
Preview 1:
The final result of the mapping from component A via intermediate component B, to target component C. These are the Travel expenses below 1500.



Preview 2:
The result of the mapping from component A to the intermediate component B, i.e. all Travel expense items. ExpRep-Target.xml is a default file name which is automatically generated because a file name was not entered in the Output XML file field.

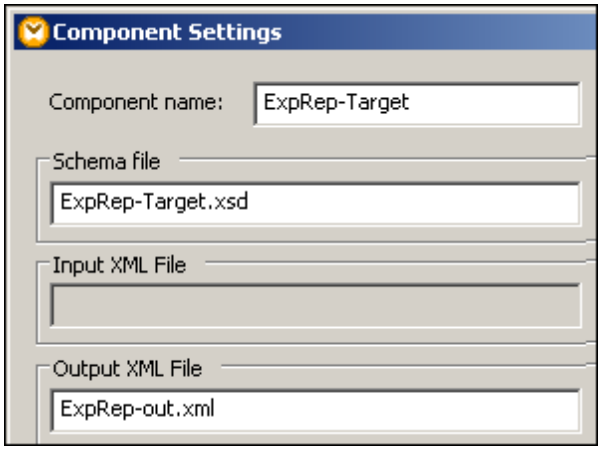


Please note:
Each mapping result is displayed in its own Preview window. Click the scroll button(s) to see the next/previous result.

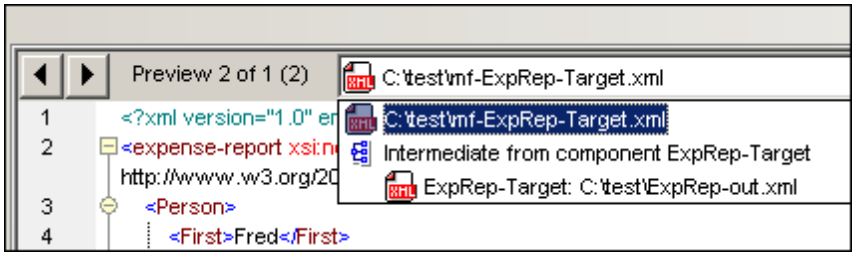


Clicking the File name combo box displays the result files(s) in a hierarchy. The final target result is shown at the top, with the intermediate result file(s) shown below. Click a file name to select it, or use the keyboard keys to navigate through the file list and press Enter.

© 2015 Altova GmbH



Setting an Output XML File in the intermediate component B, e.g. to "ExpRep-out.xml", causes the intermediate data of component B to be saved in a file of that name, e.g. ExpRep-out.xml.



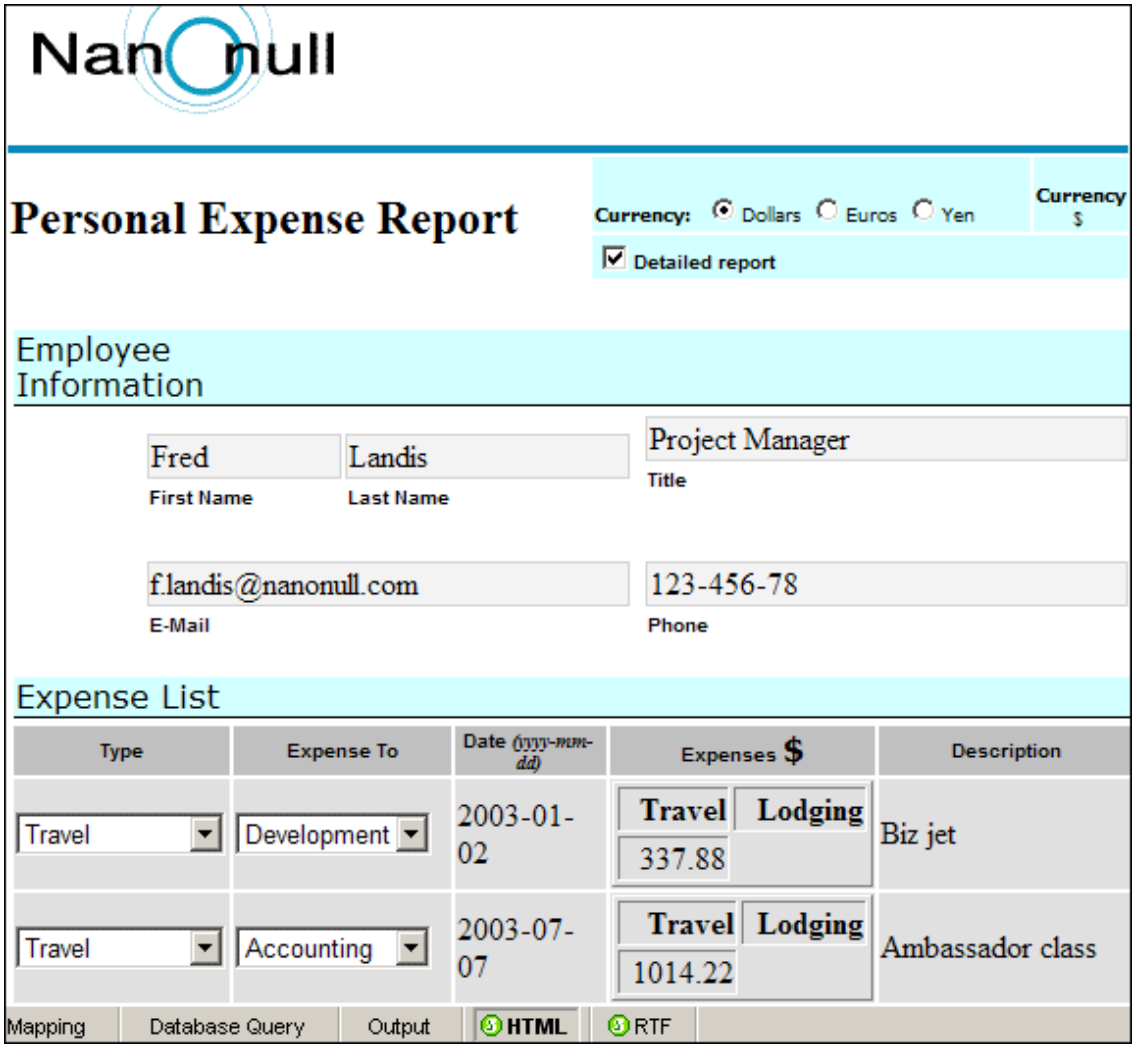
When "pass-through" is active, files created by an intermediate component are **automatically** saved as a temp files and used for further processing of that components output.

The setting "Write directly to final output file" (Tools | Options | General) determines whether the intermediate files are saved as temporary files or as physical files. For intermediate components a default file name is used to save the intermediate result, unless a dynamic file name is supplied/ mapped.

The Preview XX **of 1** means the number of final targets from the selected target component, one in this case. The Preview ... **(2**) refers to the total number of results including all intermediate components.

Displaying the result with StyleVision

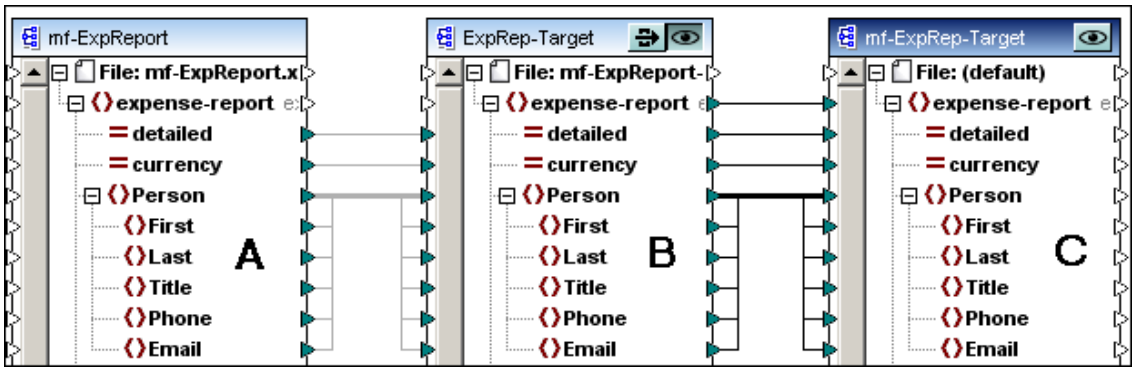
If an SPS file has been assigned to a target component, then clicking the HTML, RTF tab, will show the resulting data in the respective StyleVision tab in MapForce.



Note that only outputs of final target components of a mapping chain are shown in the StyleVision tab in MapForce. StyleVision outputs of intermediate components can not be shown.

5.3.2 Chained mappings - Pass-through inactive

The Tut-ExpReport-chain.mfd example works differently if the pass-through button is **inactive** on component B.



 © 2015 Altova GmbH

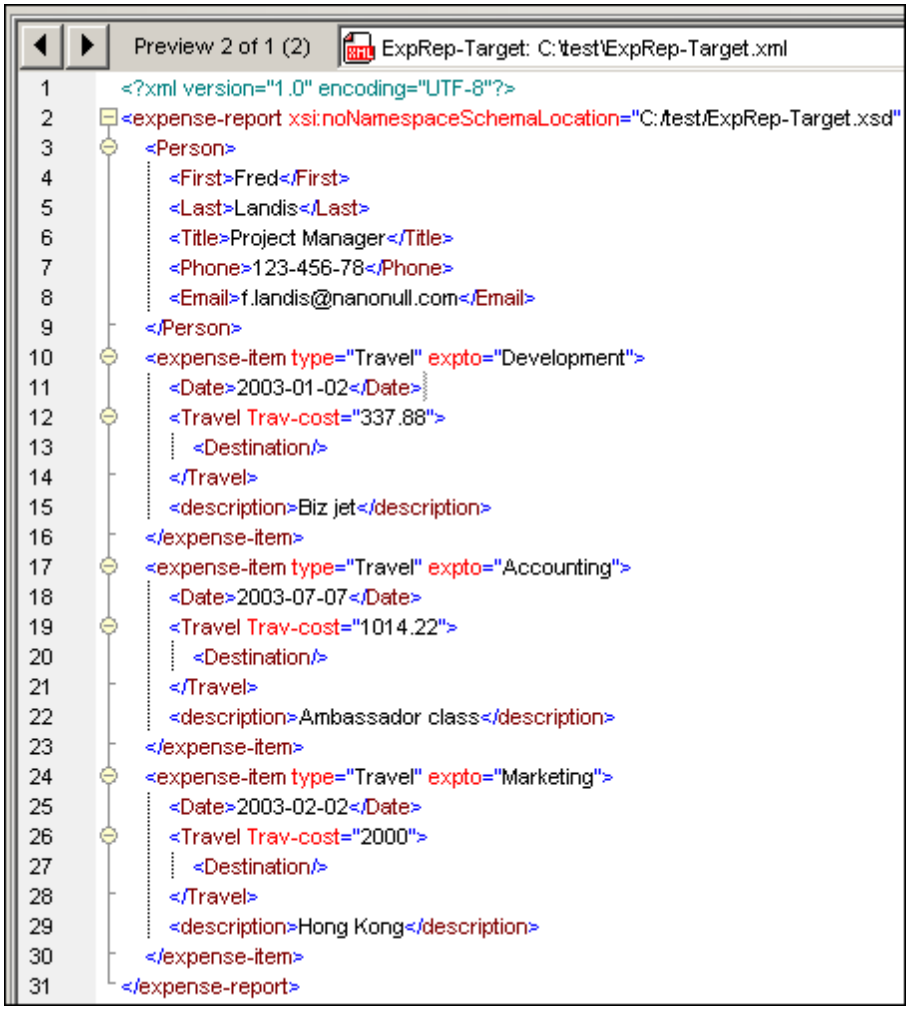
The automatic transfer of data from component A via component B and further to component C, has been interrupted by disabling the pass-through button. The Preview buttons of components B and C determine which part of the mapping chain is generated.

MapForce generates the output for the component where the Preview button is active.

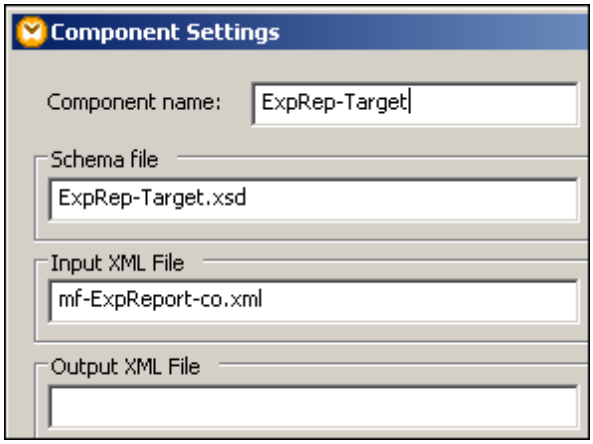
- If the Preview button of component **B** is active, then the result of mapping component A to component B is generated. Component C is ignored.

  Clicking the Output button previews the results in the Built-in execution engine.

Preview:
The result of the mapping from component A to the intermediate component B, i.e. all Travel expense items.



- If the Preview button of component **C** is active, MapForce maps the data from the intermediate component B to component C. Component A is ignored. Component B has an Input XML File, mf-ExpReport-co.xml, assigned to it, see Saving an intermediate mapping result in the text below.

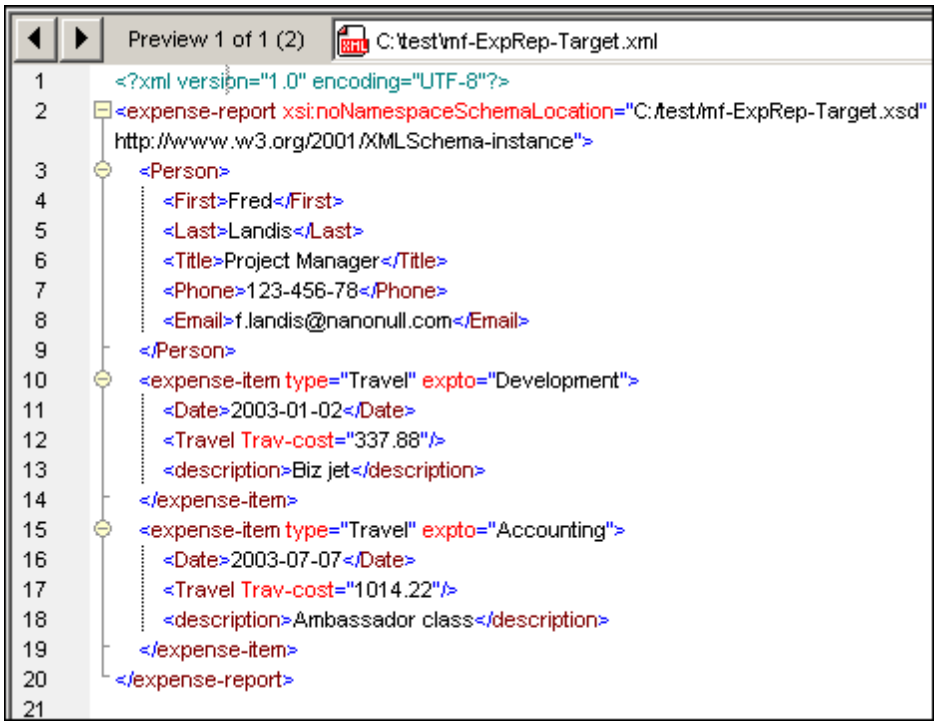


MapForce opens the intermediate file and maps its data to component C. If the input file of component B exists, this mapping will produce output. This file entry must exist here for the mapping to execute. MapForce displays an error message if the input file is missing.

When "pass-through" is inactive, the *Input XML File* field of the intermediate component is enabled, as shown above.

Note the difference to the case where component B had the "pass-through" button active, in that case the Input XML file field is automatically disabled.

Preview:
The result of the mapping from intermediate component B to the target component C, i.e. Travel expenses below 1500.



 © 2015 Altova GmbH

Note:
If this mapping is executed from the command line, or generated code, then regardless of the state of the pass-through button in component B and the selected preview component, MapForce attempts to generate the output of component B and component C. The setting of the preview button has no effect.

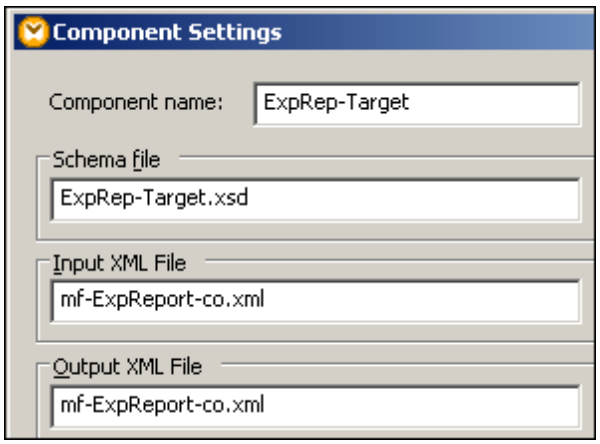
Since the Input XML file entry is different from the Output XML file entry (which is empty) the mapping chain is broken and the output for component C cannot be generated. Both the Input XML File field and Output XML File field have to be identical for code generation to succeed.

Saving the intermediate mapping result

To make the input file of the intermediate component accessible, when "pass-through" is inactive, the result of the mapping of component A to B must be saved. This file name is then placed in the *Input XML File* of component B. Only then can data be displayed in the final component C.

To save the mapping result of component B to a file:

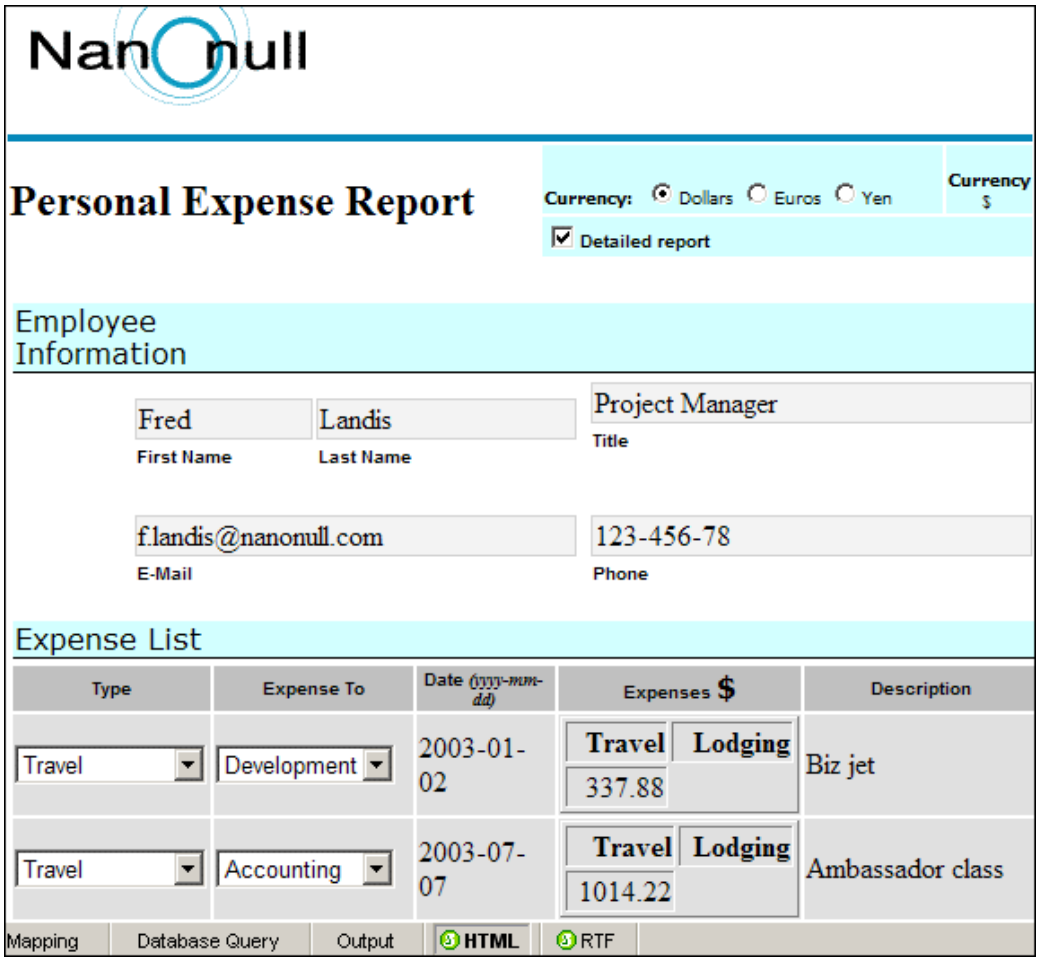
1. Click the Preview button of component B to make it active, then click the Output button.
2. Click the **Save generated output** button in the Output Preview tool bar and give the XML file a name, e.g. `mf-ExpReport-co.xml`.
3. Double click the header of component B to open the **Component Settings** dialog box, and copy the file name into the *Input XML File* field and click **OK**.



Please note: Both the Input and Output file names **must** be identical (and present) for **code generation** and execution from the command line to occur.

Displaying the result with StyleVision

If an SPS file has been assigned to a target component, then clicking the HTML, RTF tab, will show the resulting data in the respective StyleVision tab in MapForce.



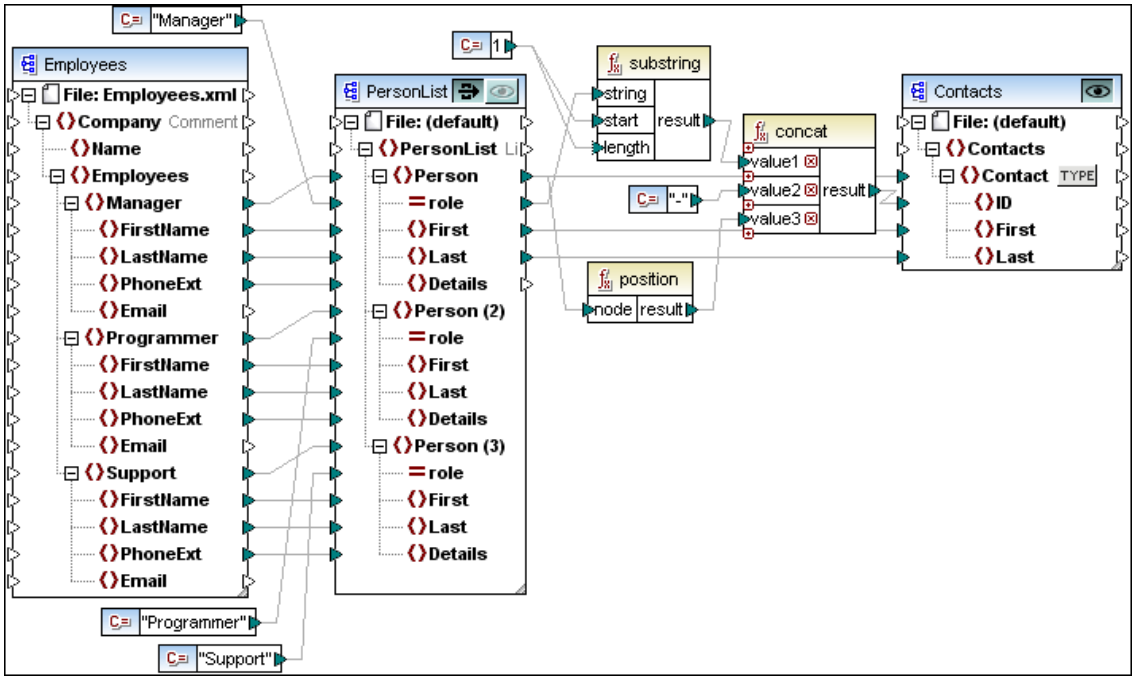
5.3.3 Chained mapping example

The example shown below is available as **ChainedPersonList.mfd** in the ...\MapForceExamples folder.

Aim:
To create two sets of employee documents, one for human resources and the other for bookkeeping.

- The document for the bookkeeping department assigns an unique ID to the employee.
- The document for the HR department has the person details, and additionally the telephone extension.

 © 2015 Altova GmbH



Components:

Employees:
The Employees.xml instance file contains four people with the roles in the sequence: manager, programmer and support.

PersonList: (output will be the HR document) - Intermediate component

- A **role** attribute is added to the person data, and the position hierarchy that exists in the Employees component is removed.
- The "pass-through" button is active.

Contacts: (output will be the bookkeeping document)

- An ID element is added to the Contact data to make sure that person data is unique.

How it works:

PersonList:

- The person element is **duplicated** twice to allow for the three types of roles that exist within the company.
- The **role** names are added as strings, using constant components, in the same sequence as in the Employees component.

Contacts:

- The **substring** function splits off the first character of the role attribute and forwards it to the concat function.
- The **position** function iterates over all the Person nodes, assigns a sequential number (starting at 1) and forwards it to the concat function.
- The **concat** function combines the substring character, a hyphen (from a constant component) and the position number and forwards it to the ID element of the Contacts

component.

Result:

PersonList component: (output: HR document)

Preview 2 of 1 (2) PersonList: C:\Documents and Settir

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <PersonList xsi:noNamespaceSchemaLocation="C:/DOCUME~1
   http://www.w3.org/2001/XMLSchema-instance">
3    <Person role="Manager">
4      <First>Vernon</First>
5      <Last>Callaby</Last>
6      <Details>582</Details>
7    </Person>
8    <Person role="Programmer">
9      <First>Frank</First>
10     <Last>Further</Last>
11     <Details>471</Details>
12   </Person>
13   <Person role="Support">
14     <First>Loby</First>
15     <Last>Matise</Last>
16     <Details>963</Details>
17   </Person>
18   <Person role="Support">
19     <First>Susi</First>
20     <Last>Sanna</Last>
21     <Details>753</Details>
22   </Person>
23 </PersonList>
```

Contacts component: (output: bookkeeping document)

Preview 1 of 1 (2) C:\Documents and Settings\ply\My

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Contacts xsi:noNamespaceSchemaLocation="C:/DOCUME~1
   http://www.w3.org/2001/XMLSchema-instance">
3    <Contact>
4      <ID>M-1</ID>
5      <First>Vernon</First>
6      <Last>Callaby</Last>
7    </Contact>
8    <Contact>
9      <ID>P-2</ID>
10     <First>Frank</First>
11     <Last>Further</Last>
12   </Contact>
13   <Contact>
14     <ID>S-3</ID>
15     <First>Loby</First>
16     <Last>Matise</Last>
17   </Contact>
18   <Contact>
19     <ID>S-4</ID>
20     <First>Susi</First>
21     <Last>Sanna</Last>
22   </Contact>
23 </Contacts>
```


© 2015 Altova GmbH

5.4 Processing Multiple Input or Output Files Dynamically

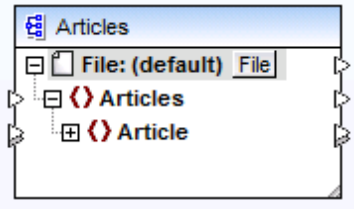
You can configure MapForce to process multiple files (for example, all files in a directory) when the mapping runs. Using this feature, you can solve tasks such as:

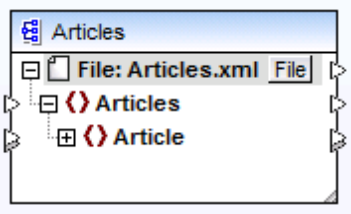
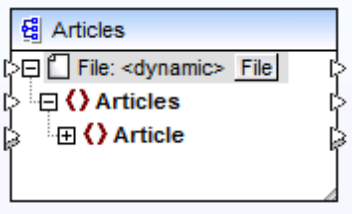
- Supply to the mapping a list of input files to be processed
- Generate as mapping output a list of files instead of a single output file
- Generate a mapping application where both the input and output file names are defined at runtime
- Convert a set of files to another format
- Split a large file into smaller parts
- Merge multiple files into one large file

You can configure a MapForce component to process multiple files in one of the following ways:

- Supply the path to the required input or output file(s) using wildcard characters instead of a fixed file name, in the component settings (see Changing the Component Settings). Namely, you can enter the wildcards * and ? in the Component Settings dialog box, so that MapForce resolves the corresponding path when the mapping runs.
- Connect to the root node of a component a sequence which supplies the path dynamically (for example, the result of the `replace-fileext` function). When the mapping runs, MapForce will read dynamically all the input files or generate dynamically all the output files.

Depending on what you want to achieve, you can use either one or both of these approaches on the same mapping. However, it is not meaningful to use both approaches at the same time on the same component. To instruct MapForce which approach you want to use for a particular component, click the **File** (File) or **File/String** (File/String) button available next to the root node of a component. This button enables you to specify the following behavior:

Use File Names from Component Settings	If the component should process one or several instance files, this option instructs MapForce to process the file name(s) defined in the Component Settings dialog box. If you select this option, the root node does not have an input connector, as it is not meaningful.  If you did not specify yet any input or output files in the Component Settings dialog box, the name of the root node is **File: (default)**. Otherwise, the root node displays the name of the input file, followed by

	a semi-colon (;), followed by the name of the output file. If the name of the input is the same with that of the output file, it is displayed as name of the root node.  Note that you can select either this option or the *Use Dynamic File Names Supplied by Mapping* option.
Use Dynamic File Names Supplied by Mapping	This option instructs MapForce to process the file name(s) that you define on the mapping area, by connecting values to the root node of the component. If you select this option, the root node gets an input connector to which you can connect values that supply dynamically the file names to be processed during mapping execution. If you have defined file names in the Component Settings dialog box as well, those values are ignored. When this option is selected, the name of the root node is displayed as **File: <dynamic>**.  This option is mutually exclusive with the *Use File Names from Component Settings* option.

Multiple input or output files can be defined for the following components:

- XML files
- Text files (CSV*, FLF* files and FlexText** files)
- EDI documents**
- Excel spreadsheets**
- XBRL documents**

 © 2015 Altova GmbH

** Requires MapForce Professional Edition*
*** Requires MapForce Enterprise Edition*

The following table illustrates support for dynamic input and output file and wildcards in MapForce languages.

Target language	Dynamic input file name	Wildcard support for input file name	Dynamic output file name
XSLT 1.0	*	Not supported by XSLT 1.0	Not supported by XSLT 1.0
XSLT 2.0	*	*(1)	*
C++	*	*	*
C#	*	*	*
Java	*	*	*
BUILTIN	*	*	*

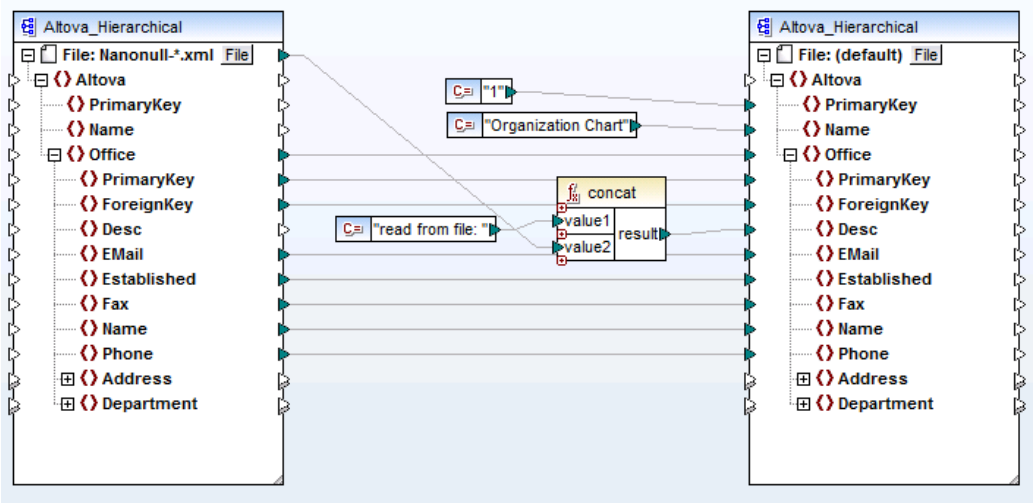
Legend:

*	Supported
(1)	Uses the **fn:collection** function. The implementation in the Altova XSLT 2.0 and XQuery engines resolves wildcards. Other engines may behave differently. For details on how to transform XSLT 1.0/2.0 code using the RaptorXML Server engine, see Generating XSLT 1.0, or 2.0 code

5.4.1 Mapping Multiple Input Files to a Single Output File

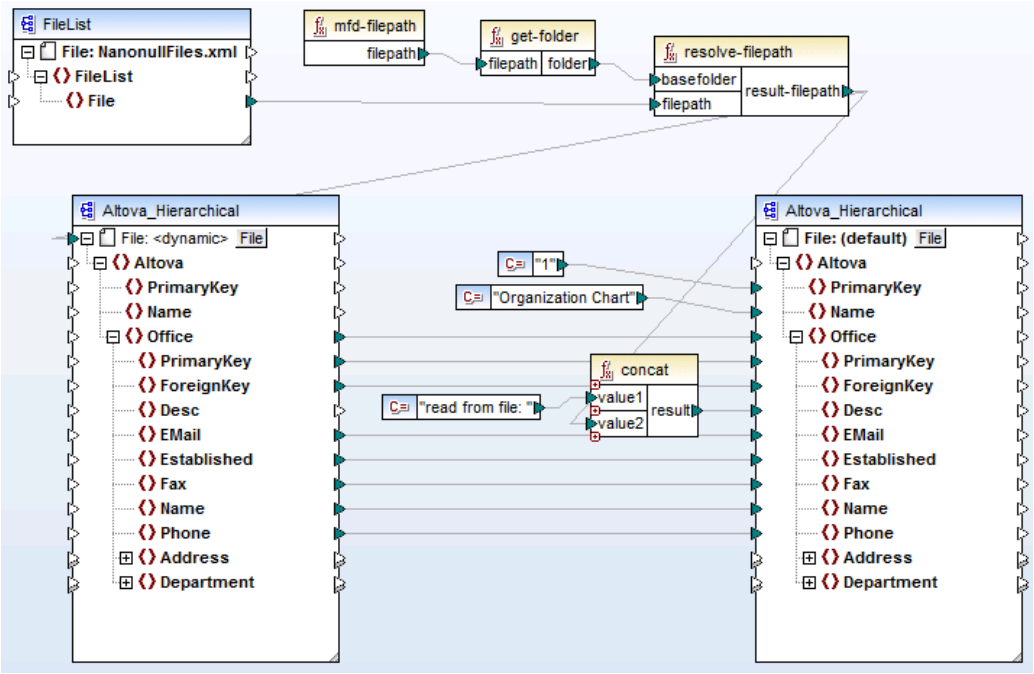
To process multiple input files, do one of the following:

- Enter a file path with wildcards (* or ?) as input file in the Component Settings dialog box. All matching files will be processed. The example below uses the ***** wildcard character in the Input XML file field to supply as mapping input all files whose name begins with "Nanonull-". Multiple input files are being merged into a **single** output file because there is no dynamic connector to the target component, while the source component accesses multiple files using the wildcard *. Notice that the name of the root node in the target component is **File: <default>**, indicating that no output file path has been defined in the Component Settings dialog box. The multiple source files are thus appended in the target document.



MergeMultipleFiles.mfd (MapForce Basic Edition)

- Map a **sequence** of strings to the *File* node of the source component. Each string in the sequence represents one file name. The strings may also contain wildcards, which are automatically resolved. A sequence of file names can be supplied by components such as an XML file.



MergeMultipleFiles_List.mfd (MapForce Basic Edition)

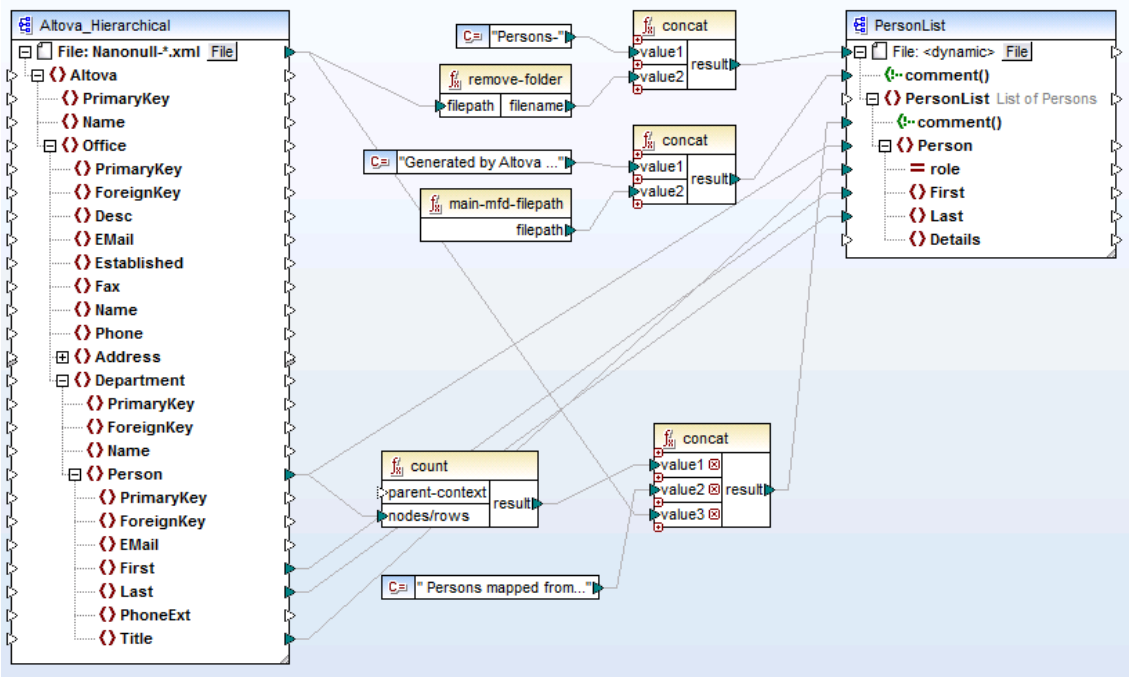
5.4.2 Mapping Multiple Input Files to Multiple Output Files

To map multiple files to multiple target files, you need to generate unique output file names. In some cases, the output file names can be derived from strings in the input data, and in other

 © 2015 Altova GmbH

cases it is useful to derive the output file name from the input file name, e.g. by changing the file extension.

In the following mapping, the output file name is derived from the input file name, by adding the prefix "Persons-" with the help of the **concat** function.



MultipleInputToMultipleOutputFiles.mfd (MapForce Basic Edition)

Note: Avoid simply connecting the input and output root nodes directly, without using any processing functions. Doing this will overwrite your input files when you run the mapping. You can change the output file names using functions such as the **concat** function, as shown above.

The menu option **File | Mapping Settings** allows you to define globally the file path settings used by the mapping (see Changing the mapping settings).

5.4.3 Supplying File Names as Mapping Parameters

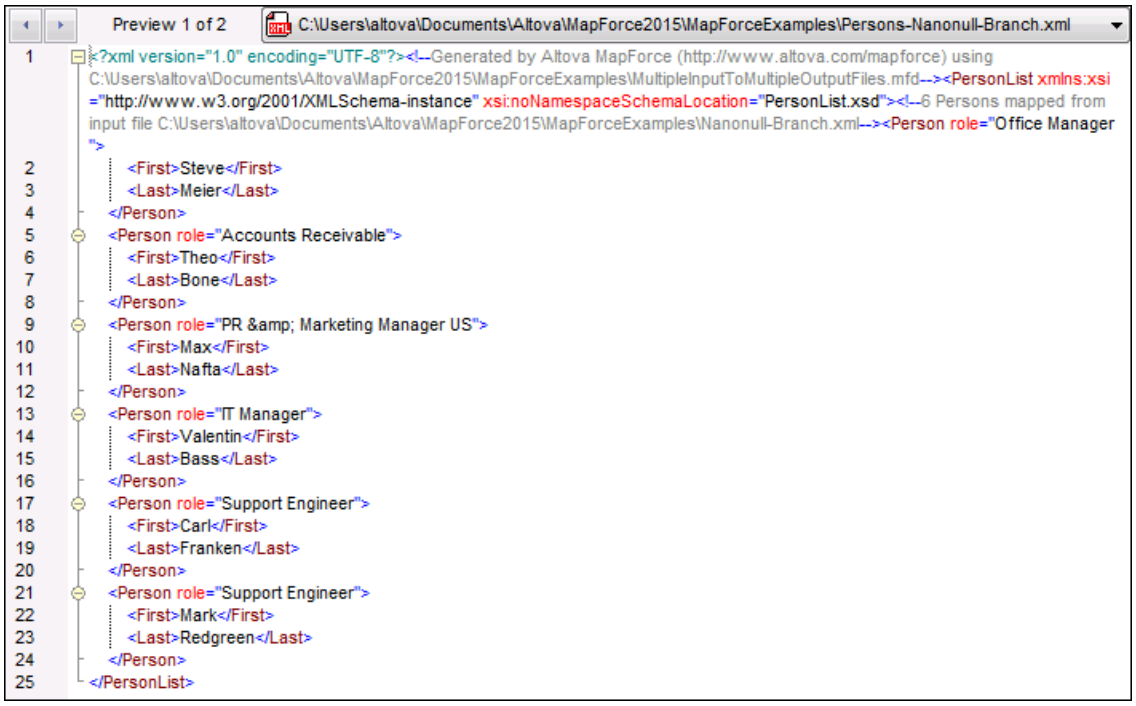
To supply custom file names as input parameters to the mapping, do the following:

1. Add an Input component to the mapping (On the **Function** menu, click **Insert Input**). For more information about such components, see Simple Input.
1. Click the **File** (File) or **File/String** (File/String) button of the source component and select **Use Dynamic File Names Supplied by Mapping**.
2. Connect the Input component to the root node of the component which acts as mapping source.

For a worked example, see Example: Using File Names as Mapping Parameters.

5.4.4 Previewing Multiple Output Files

Click the Output tab to display the mapping result in a preview window. If the mapping produces multiple output files, each file has its own numbered pane in the Output tab. Click the arrow buttons to see the individual output files.



MultipleInputToMultipleOutputFiles.mfd

To save the generated output files, do one of the following:

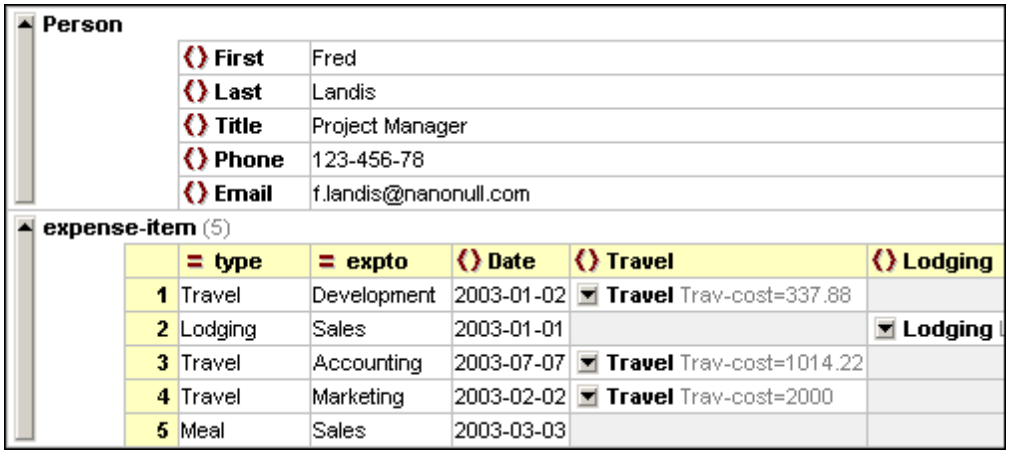
- On the **Output** menu, click **Save All Output Files** ( ).
- Click the **Save all generated outputs** ( ) toolbar button.

5.4.5 Example: Split One XML File into Many

This example shows you how to generate dynamically multiple XML files from a single source XML file. The accompanying mapping for this example is available at the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\Tut-ExpReport-dyn.mfd**.

The source XML file (available in the same folder as the mapping) consists of the expense report for a person called "Fred Landis" and contains five expense items of different types. The aim of the example is to generate a separate XML file for each of the expense items listed below.

© 2015 Altova GmbH



Person

First	Fred
Last	Landis
Title	Project Manager
Phone	123-456-78
Email	f.landis@nanonull.com

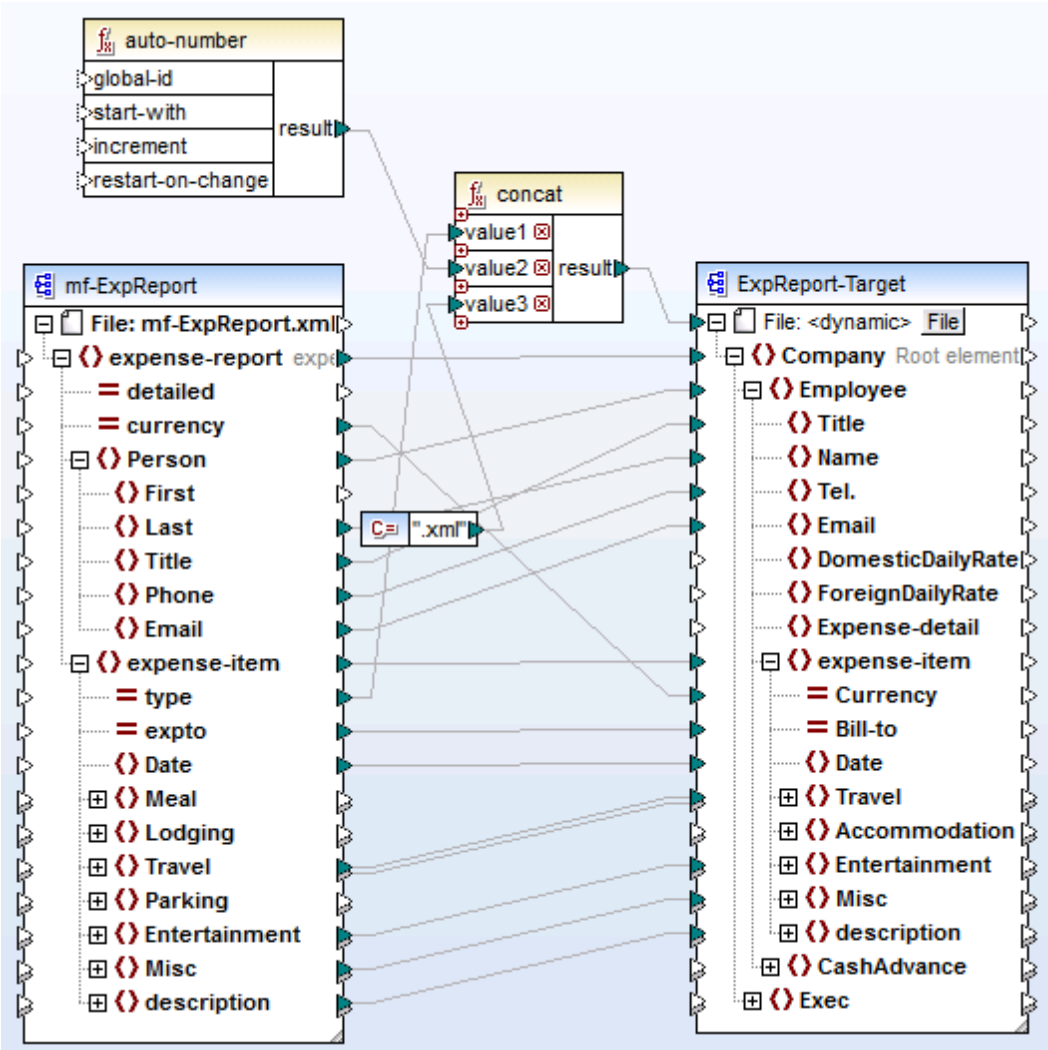
expense-item (5)

	type	expto	Date	Travel	Lodging
1	Travel	Development	2003-01-02	**Travel** Trav-cost=337.88	
2	Lodging	Sales	2003-01-01		**Lodging**
3	Travel	Accounting	2003-07-07	**Travel** Trav-cost=1014.22	
4	Travel	Marketing	2003-02-02	**Travel** Trav-cost=2000	
5	Meal	Sales	2003-03-03		

mf-ExpReport.xml (as shown in XMLSpy Grid view)

As the `type` attribute defines the specific expense item type, this is the item we will use to split up the source file. To achieve the goal of this example, do the following:

1. Insert a **`concat`** function (you can drag it from the **core | string functions** library of the Libraries pane).
2. Insert a constant (on the **Insert** menu, click **Constant**) and enter ".xml" as its value.
3. Insert the **`auto-number`** function (you can drag it from the **core | generator functions** library of the Libraries pane).
1. Click the **File** (File) or **File/String** (File/String) button of the target component and select **Use Dynamic File Names Supplied by Mapping**.
4. Create the connections as shown below and then click the **Output** tab to see the result of the mapping.



Tut-ExpReport-dyn.mfd (MapForce Basic Edition)

Note that the resulting output files are named dynamically as follows:

- The `type` attribute supplies the first part of the file name (for example, "Travel").
- The **`auto-number`** function supplies the sequential number of the file (for example, "Travel1", "Travel2", and so on).
- The constant supplies the file extension, which is ".xml", thus "Travel1.xml" is the file name of the first file.

 © 2015 Altova GmbH

5.5 Supplying Parameters to the Mapping

You can pass simple values to a mapping by means of simple input components. On the mapping area, simple input components play the role of a source component which has a simple data type (for example, string, integer, and so on) instead of a structure of items and sequences. Consequently, you can create a simple input component instead of (or in addition to) a file-based source component.In the generated XSLT file, simple input components correspond to stylesheet parameters.

You can create each simple input component (or parameter) as optional or mandatory (see Input Component Settings). If necessary, you can also create default values for the mapping input parameters (see Creating a Default Input Value). This enables you to safely run the mapping even if you do not explicitly supply a input parameter value at mapping execution time.

Input parameters added on the main mapping area should not be confused with input parameters in user-defined functions (see User-defined functions). There are some similarities and differences between the two, as follows.

Input parameters on the mapping	Input parameters of user-defined functions
Added from **Function | Insert Input** menu.	Added from **Function | Insert Input** menu.
Can have simple data types (string, integer, and so on).	Can have simple as well as complex data types.
Applicable to the entire mapping.	Applicable only in the context of the function in which they were defined.

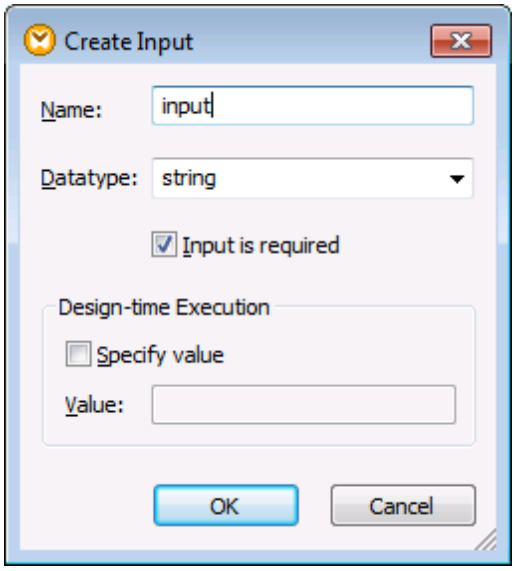
When you create a reversed mapping (using the menu command **Tools | Create Reversed Mapping**), a simple input component becomes a simple output component.

For an example, see Example: Using File Names as Mapping Parameters.

5.5.1 Adding Simple Input Components

To add a simple input to the mapping:

1. Make sure that the mapping window displays the main mapping (not a user-defined function).
2. On the **Function** menu, click **Input**.
3. Enter a name and select the data type required for this input. If the input should be treated as a mandatory mapping parameter, select the **Input is required** check box. For a complete list of settings, see Simple Input Component Settings.
4. Click **OK**.

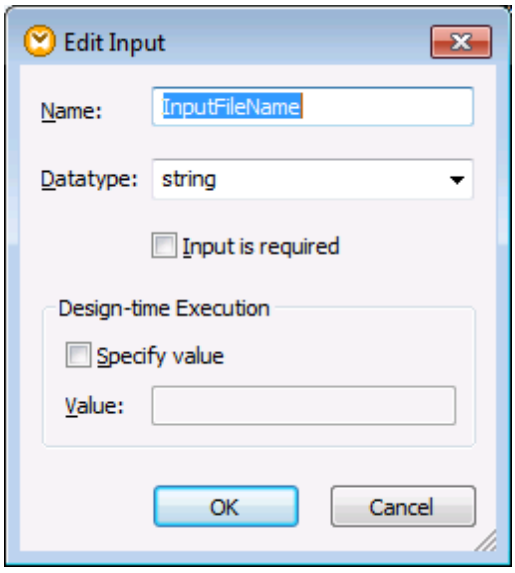


Create Input dialog box

You can change later any of the settings defined here (see Simple Input Component Settings).

5.5.2 Simple Input Component Settings

You can define the settings applicable to a simple input component either when adding it to the mapping area. You can also change the setting at a later time, from the Edit Input dialog box.



Edit Input dialog box

 © 2015 Altova GmbH

To open the Edit Input dialog box, do one of the following:

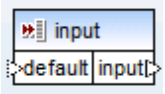
- Select the component, and, on the **Component** menu, click **Properties**.
- Double-click the component.
- Right-click the component, and then click **Properties**.

The available settings are as follows.

Name	Enter a descriptive name for the input parameter corresponding to this component. At mapping execution time, the value entered in this text box becomes the name of the parameter supplied to the mapping; therefore, no spaces or special characters are allowed.
Datatype	By default, any input parameter is treated as string data type. If the parameter should have a different data type, select the respective value from the list. When the mapping is executed, MapForce casts the input parameter to the data type selected here.
Input is required	When enabled, this setting makes the input parameter mandatory (that is, the mapping cannot be executed unless you supply a parameter value). Disable this check box if you want to specify a default value for the input parameter (see Creating a Default Input Value).
Specify value	This setting is applicable only if you execute the mapping during design time, by clicking the **Preview** tab. It allows you to enter directly in the component the value to use as mapping input.
Value	This setting is applicable only if you execute the mapping during design time, by clicking the **Preview** tab. To enter a value to be used by MapForce as mapping input, select the Specify Value check box, and then type the required value.

5.5.3 Creating a Default Input Value

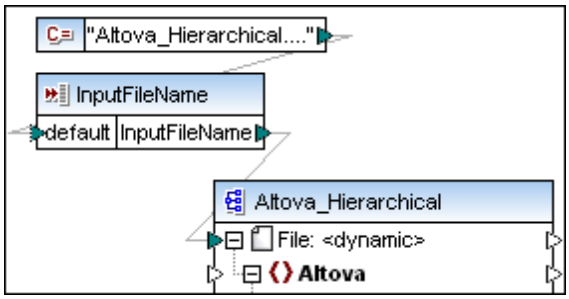
After you add an Input component to the mapping area, notice the **default** item to the left of the component.



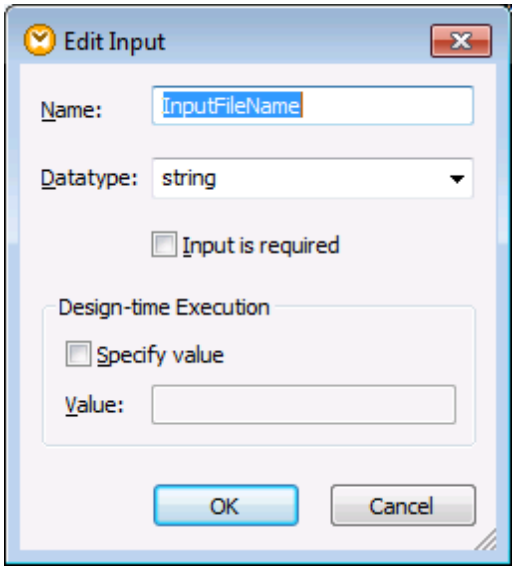
Simple input component

The default item enables you to connect an optional default value to this input component, as follows:

1. Add a constant component (on the **Insert** menu, click **Constant**), and then connect it to the **default** item of the input component.



2. Double click the input component and make sure that the **Input is required** check box is disabled. When you create a default input value, this setting is not meaningful and causes mapping validation warnings.



3. Click **OK**.

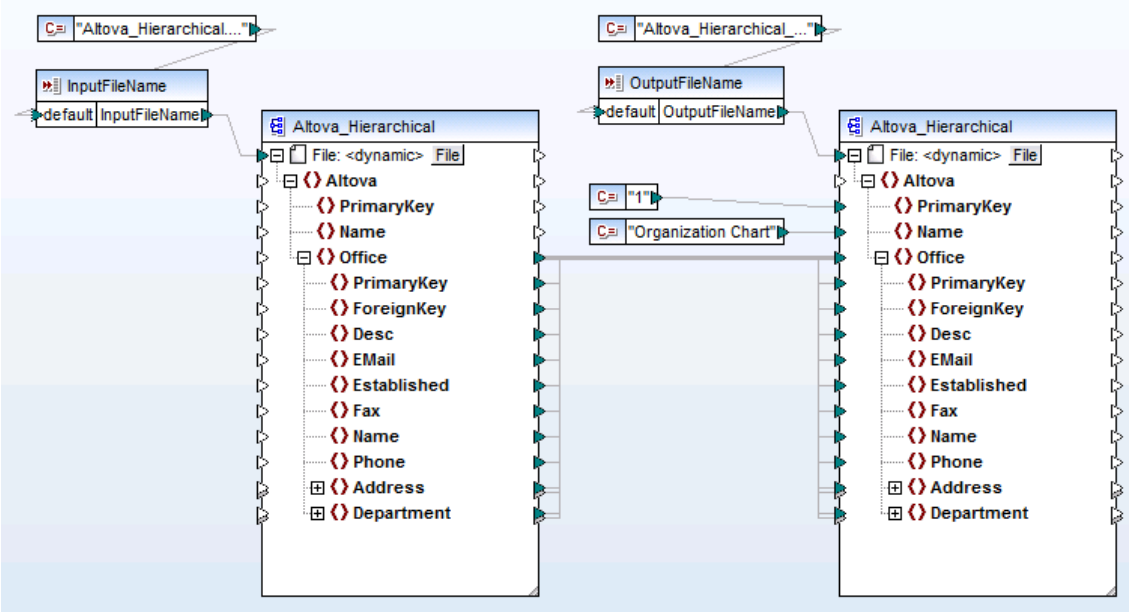
Note: If you click the **Specify value** check box and enter a value in the adjacent box, the entered value takes precedence over the default value when you preview the mapping (that is, at design-time execution). However, the same value has no effect in the generated code.

5.5.4 Example: Using File Names as Mapping Parameters

This example walks you through the steps required to execute a mapping that takes input parameters at runtime. The mapping design file used in this example is available at the following path: **<Documents>\Altova\MapForce2016\MapForceExamples\FileNamesAsParameters.mfd**.

The mapping uses two input components: **InputFileName** and **OutputFileName**. These supply the input file name (and the output file name, respectively) of the source and target XML file. For this reason, they are connected to the **File: <dynamic>** item.

 © 2015 Altova GmbH



FileNamesAsParameters.mfd (MapForce Basic Edition)

Both the **InputFileName** and **OutputFileName** components are simple input components in the mapping, so you can supply them as input parameters when executing the mapping. The following sections illustrate how to do this in the following transformation languages:

- XSLT 2.0, using RaptorXML Server

XSLT 2.0

If you generate code in XSLT 1.0 or XSLT 2.0, the input parameters are written to the **DoTransform.bat** batch file, for execution by RaptorXML Server (see RaptorXML Server). To use a different input (or output) file, you can either pass the required parameters at command line, when calling the **DoTransform.bat** file, or edit the latter to include the required parameters.

To supply a custom input parameter in the **DoTransform.bat** file:

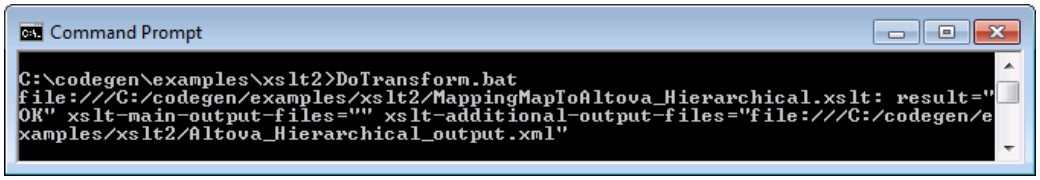
1. Generate the XSLT 2.0 code (**File | Generate Code In | XSLT 2.0**) from the **FileNamesAsParameters.mfd** sample.
2. Copy the **Altova_Hierarchical.xml** file from the **<Documents>\Altova\MapForce2016\MapForceExamples** directory to the directory where you generated the XSLT 2.0 code (in this example, **c:\codegen\examples\xslt2**). This file will act as custom parameter.
3. Edit **DoTransform.bat** to include the custom input parameter either before or after %* (as highlighted below). Note that the parameter value is enclosed with single quotes. The available input parameters are listed in the **rem** (Remark) section.

```
@echo off

RaptorXML xslt --xslt-version=2 --
input="MappingMapToAltova_Hierarchical.xslt" --
```

```
param=InputFileName:'Altova_Hierarchical.xml' %*
"MappingMapToAltova_Hierarchical.xslt"
rem --param=InputFileName:
rem --param=OutputFileName:
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

When you run the DoTransform.bat file, RaptorXML Server completes the transformation using **Altova_Hierarchical.xml** as input parameter.



 © 2015 Altova GmbH

5.6 Returning String Values from a Mapping

Use a simple output component when you need to return a string value from the mapping. On the mapping area, simple output components play the role of a target component which has a string data type instead of a structure of items and sequences. Consequently, you can create a simple output component instead of (or in addition to) a file-based target component. For example, you can use a simple output component to quickly test and preview the output of a function (see Example: Testing Function Output).

Simple output components should not be confused with output parameters of user-defined functions (see User-defined functions). There are some similarities and differences between the two, as follows.

Output components	Output parameters of user-defined functions
Added from **Function | Insert Output** menu.	Added from **Function | Insert Output** menu.
Have "string" as data type.	Can have simple as well as complex data types.
Applicable to the entire mapping.	Applicable only in the context of the function in which they were defined.

If necessary, you can add multiple simple output components to a mapping. You can also use simple output components in combination with file-based target components. When your mapping contains multiple target components, you can preview the data returned by a particular component by clicking the **Preview** ( ) button in the component title bar, and then clicking the **Output** tab on the Mapping window.

You can use simple output components as follows in MapForce transformation languages:

Language	How it works
XSLT 1.0, XSLT 2.0	If the generated XSLT files, a simple output components defined in the mapping becomes the output of the XSLT transformation. If you are using RaptorXML Server, you can instruct RaptorXML Server to write the mapping output to the file passed as value to the `--output` parameter. To write the output to a file, add or edit to the `--output` parameter in the **DoTransform.bat** file. For example, the following **DoTransform.bat** file has been edited to write the mapping output to the **Output.txt** file (see highlighted text). `RaptorXML xslt --xslt-version=2 --input="MappingMapToResult1.xslt" --output="Output.txt" %* "MappingMapToResult1.xslt"`

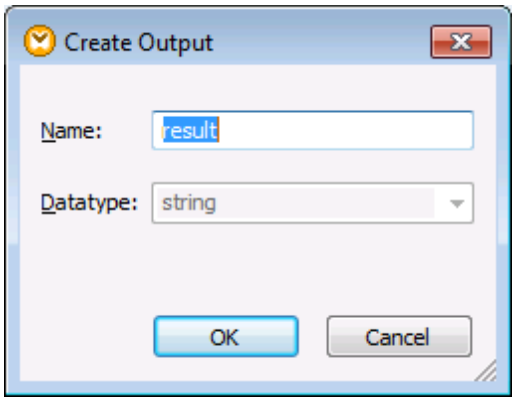
	If an `--output` parameter is not defined, the mapping output will be written to the standard output stream (stdout) when the mapping is executed.

When you create a reversed mapping (using the menu command **Tools | Create Reversed Mapping**), the simple output component becomes a simple input component.

5.6.1 Adding Simple Output Components

To add an Output component to the mapping area:

1. Make sure that the mapping window displays the main mapping (not a user-defined function).
2. On the **Function** menu, click **Output**.
3. Enter a name for the component.
4. Click **OK**.



Create Output dialog box

You can change the component name at any time later, in one of the following ways:

- Select the component, and, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.

5.6.2 Example: Previewing Function Output

This example illustrates how to preview the output returned by MapForce functions with the help of simple output components. You will make the most of this example if you already have a basic understanding of functions in general, and of MapForce functions in particular. If you are new to MapForce functions, you may want to refer to Using Functions before continuing.

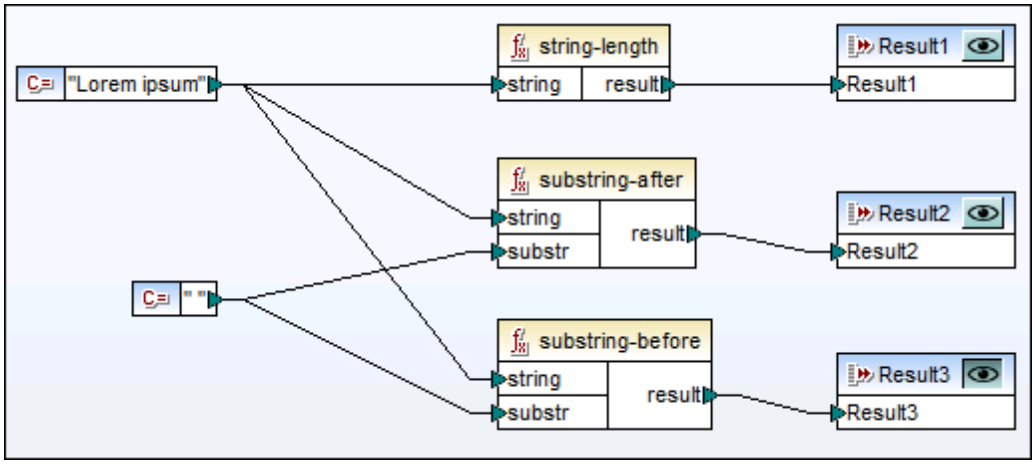
Our aim is to add a number of functions to the mapping area, and learn how to preview their output with the help of simple output components. In particular, the example uses a few simple functions available in the core library. Here is a summary of their usage:

 © 2015 Altova GmbH

string-length	Returns the number of characters in the string provided as argument. For example, if you pass to this function the value "Lorem ipsum", the result is "11", since this is the number of characters that the text "Lorem ipsum" takes.
substring-after	Returns the part of the string that occurs after the separator provided as argument. For example, if you pass to this function the value "Lorem ipsum" and the space character (" "), the result is "ipsum".
substring-before	Returns the part of the string that occurs before the separator provided as argument. For example, if you pass to this function the value "Lorem ipsum" and the space character (" "), the result is "Lorem".

To test each of these functions against a custom text value ("Lorem ipsum", in this example), follow the steps below:

1. Add a constant with the value "Lorem ipsum" to the mapping area (use the menu command **Insert | Constant**). The constant will be the input parameter for each of the functions to be tested.
2. Add the **string-length**, **substring-after**, and **substring-before** functions to the mapping area, by dragging them to the mapping area from the core library, **string functions** section.
3. Add a constant with an empty space (" ") as value. This will be the separator parameter required by the **substring-after** and **substring-before** functions.
4. Add three simple output components (use the menu command **Function | Insert Output**). In this example, they have been named *Result1*, *Result2*, and *Result3*, although you can give them another title.
5. Connect the components as illustrated below.



Testing function output with simple output components

As shown in the sample above, the "Lorem ipsum" string acts as input parameter to each of the **string-length**, **substring-after**, and **substring-before** functions. In addition to this, the the **substring-after** and **substring-before** functions take a space value as second input parameter. The *Result1*, *Result2*, and *Result3* components can be used to preview the result of each function.

To preview the output of any function

- Click the **Preview** ( ) button in the component title bar, and then click the **Output** tab on the Mapping window.

© 2015 Altova GmbH

5.7 Using Variables

Intermediate variables are a special type of component used to solve various advanced mapping problems. They store an intermediate mapping result for further processing.

- Variables work in all languages except XSLT1.0.
- Variable results are always sequences, i.e. a delimited list of values, and can also be used to create sequences.
- Variables are structural components, with a root node, and do not have instances (XML files etc.) associated to them.
- Variables make it possible to compare items of one sequence, to other items within the same sequence.
- Variables can be used to build intermediate sequences. Records can be filtered before passing them on to a target, or filtered after the variable by using the position function for example.

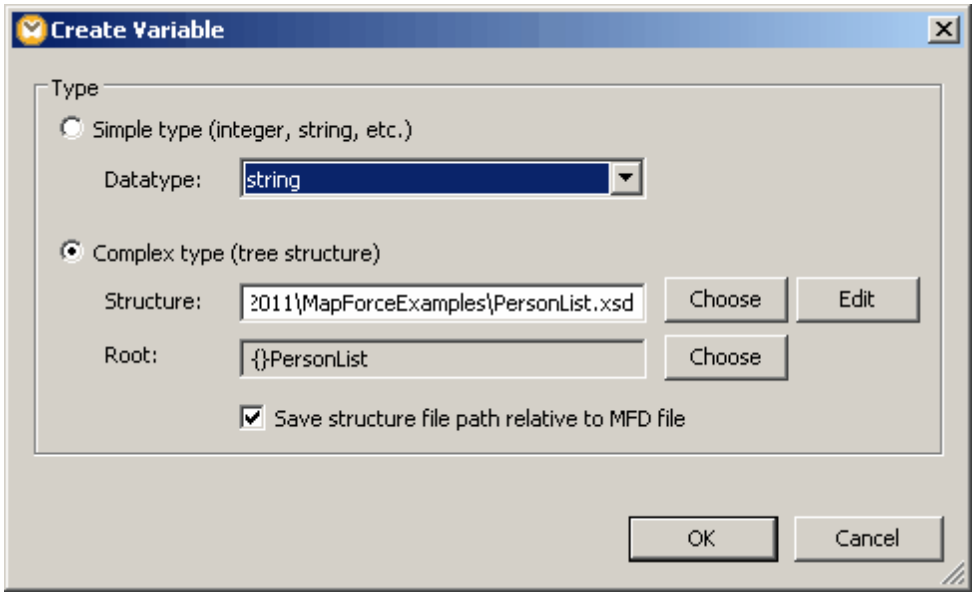
The following table outlines differences between variables and chained mappings.

Chained mappings	Variables
Involve two totally independent steps.	Evaluated depending on context / scope. Controlled by "compute-when" connection
Intermediate results are stored externally in XML files when mapping is executed.	Intermediate results are stored internally, not in any physical files, when mapping is executed.
Intermediate result can be previewed using preview button.	Variable result cannot be previewed.

Inserting intermediate variables

There are several ways of inserting intermediate variables: Using the menu option, by clicking the Var. icon, or by right clicking input/output icons and creating variables based on the input/output components.

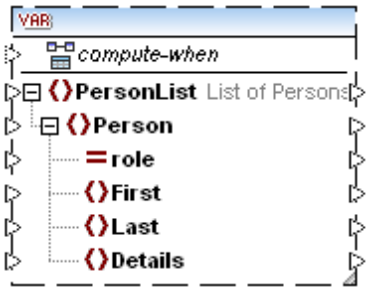
1. Select **Insert | Variable** or click the Variable icon in the icon bar. You can now select if you want to insert a simple or complex variable.



2. Click the radio button for the type of variable you want to insert, i.e. Simple type, or Complex type.

   If you clicked the "Complex type" radio button:
3. Click the "Choose" button to select XML Schema for example, and select the Root item from the next dialog box.
4. Click OK to insert the variable.

Complex variable:



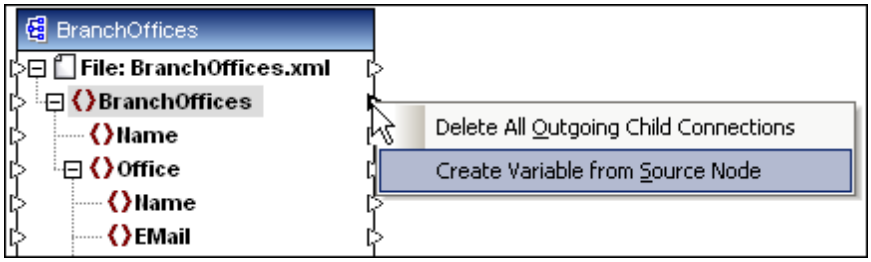
Simple variable:

Have a single mappable item/value e.g. string, integer. Note that the "value" item can be duplicated.

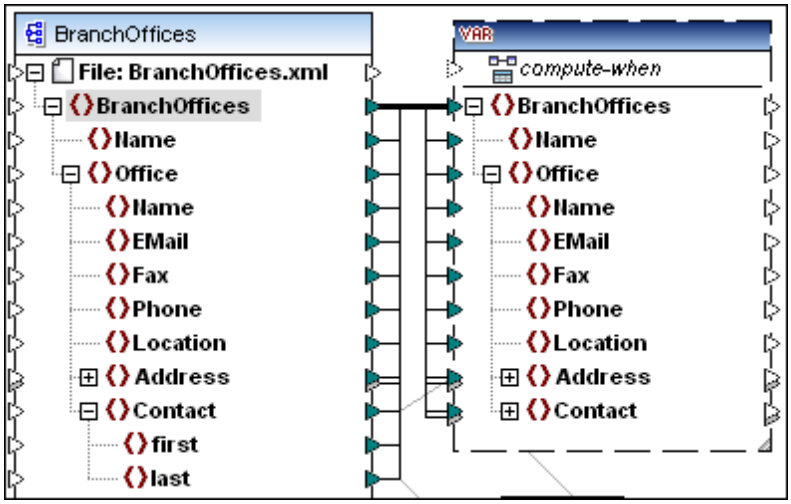
Alternate methods of inserting variables:

- Right click an **output** icon of a component (e.g. BranchOffices) and select "Create Variable from Source node".

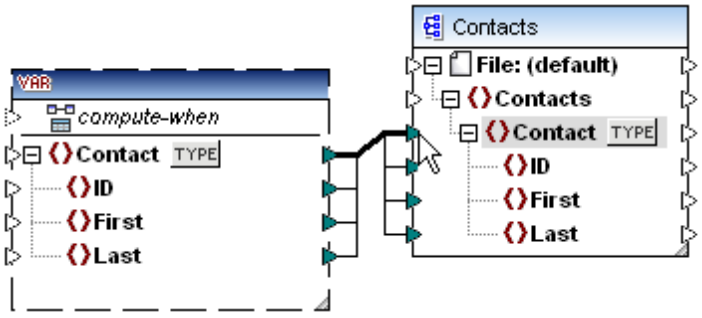
 © 2015 Altova GmbH



This creates a complex variable using the same source schema and automatically connects all items with a copy-all connection.

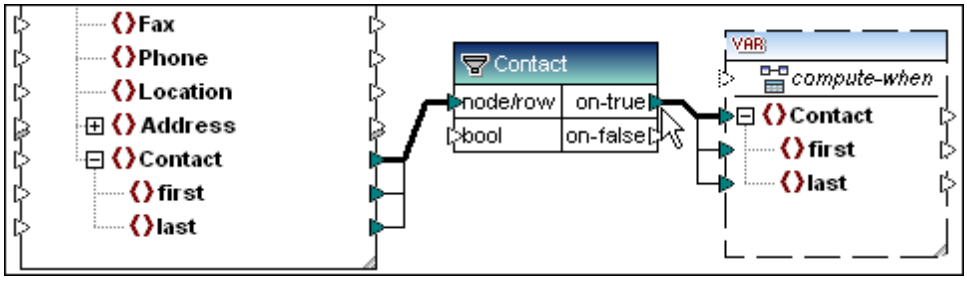


- Right click an **input** icon of a target component (e.g. Contact) and select "Create Variable for Target Node".



This creates a complex variable using the same schema as the target, with the Contact item as the root node, and automatically connects all items with a copy-all connection.

- Right click an **output** icon of a filter component (on-true/on-false) and select "Create Variable from Source node".



This creates a complex component using the source schema, and automatically uses the item linked to the filter input node/row, i.e. Contact, as the root element of the intermediate component.

Compute-when

The compute-when input item allows you to control the **scope** of the variable; in other words when and how often the variable value is computed when the mapping is executed. You do not have to connect this input in many cases, but it can be essential to override the default context, or to optimize mapping performance.

A **subtree** in the following text means the set of an item/node in a target component and all of its descendants, e.g. a <Person> element with its <FirstName> and <LastName> child elements.

Variable value means the data that is available at the **output** side of the variable component.

- For simple variables, it is a sequence of atomic values that have the datatype specified in the component properties.
- For complex variables, it is a sequence of root nodes (of the type specified in the component properties), each one including all its descendant nodes.

The sequence of atomic values (or nodes) may contain one or even zero elements. This depends on what is connected to the input side of the variable component, and to any parent items in the source and target components.

Compute-when not connected (default)

If the compute-when input item is **not connected** (to an output node of a source component), the variable value is computed whenever it is **first** used in a **target** subtree (via a connector from the variable component directly to a node in the target component, or indirectly via functions). The same variable value is also used for all target child nodes inside the subtree.

The actual variable value depends on any connections between parent items of the source and target components.

This default behavior is the same as that of complex outputs of regular user-defined functions and Web service function calls.

If the variable output is connected to multiple **unrelated** target nodes, the variable value is computed separately for each of them. This can produce different results in each case, because different parent connections influence the context in which the variable's value is evaluated.

 © 2015 Altova GmbH

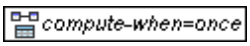
Compute-when - connected

By connecting compute-when to an **output node** of a source component, the variable is computed whenever that **source item** is **first** used in a target subtree.

The variable actually acts as if it were a child item of the item connected to compute-when.

This makes it possible to **bind** the variable to a specific source item, i.e. at runtime the variable is re-evaluated whenever a new item is read from the sequence in the source component.

This relates to the general rule governing connections in MapForce - "for each source item, create one target item". With compute-when, it means "for each source item, compute the variable value".

Compute-once 

Right clicking the "compute-when" icon and selecting "Compute Once" from the context menu changes the icon to "compute-when=once" and also removes the input icon.

This setting causes the variable value to be computed once **before** each of the target components, making the variable essentially a global constant for the rest of the mapping.

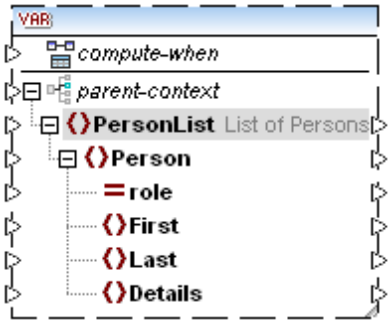
In a user-defined function, the variable is evaluated each time the function is called, before the actual function result is evaluated.

Parent-context

The main use of adding a parent-context is when using multiple filters and you need an additional parent node to iterate over.

To add a parent-context to a variable:

- Right click the root node, e.g. PersonList and select "Add Parent Context" from the context menu.
  This adds a new node, "parent-context", to the existing hierarchy.

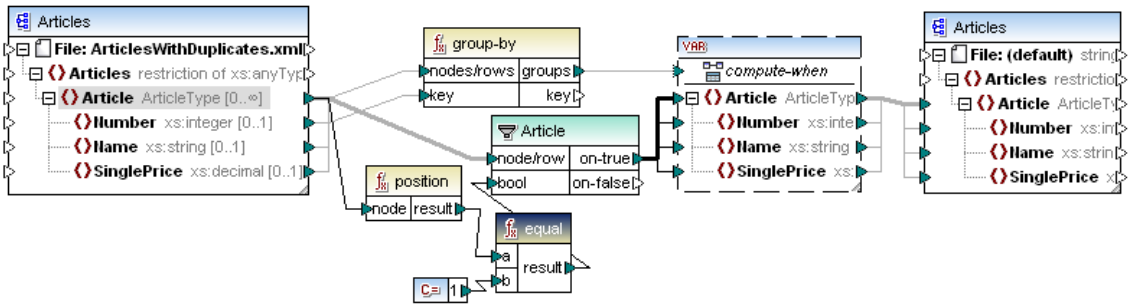


The parent context adds a virtual "parent" node to the hierarchy within the component. This allows you to iterate over an additional node in the same, or different source component.

5.7.1 Variables - use cases

Filtering out multiple instances of the same record from a source instance

Source data can often contain multiple instances of the same record. In most cases you would only want one of these records to be mapped to the target component. The example below is available as **DistinctArticles.mfd** in the ...\MapForceExamples folder.



The ArticlesWithDuplicates.xml file contains two articles both having the same article number (two with article no. 1 and two with article no 3).

Articles

= xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance	
= xsi:noNamespac...	Articles.xsd	

Article (6)

	Number	Name	SinglePrice
1	1	T-Shirt	25
2	1	T-Shirt Duplicate	25
3	2	Socks	2.30
4	3	Pants	34
5	4	Jacket	57.50
6	3	Pants Duplicate	34

The article Number is used as the key in the **group-by** function, so it creates one group per unique article number. Each group thus contains one article and all the unwanted duplicates of that article. The next step is to extract the first item of each group and discard the rest.

The connection of the group output to "compute-when" causes the variable to be evaluated once for each group, in the context of that group. This establishes an additional context level, as if we had connected a parent element of the target Article element.

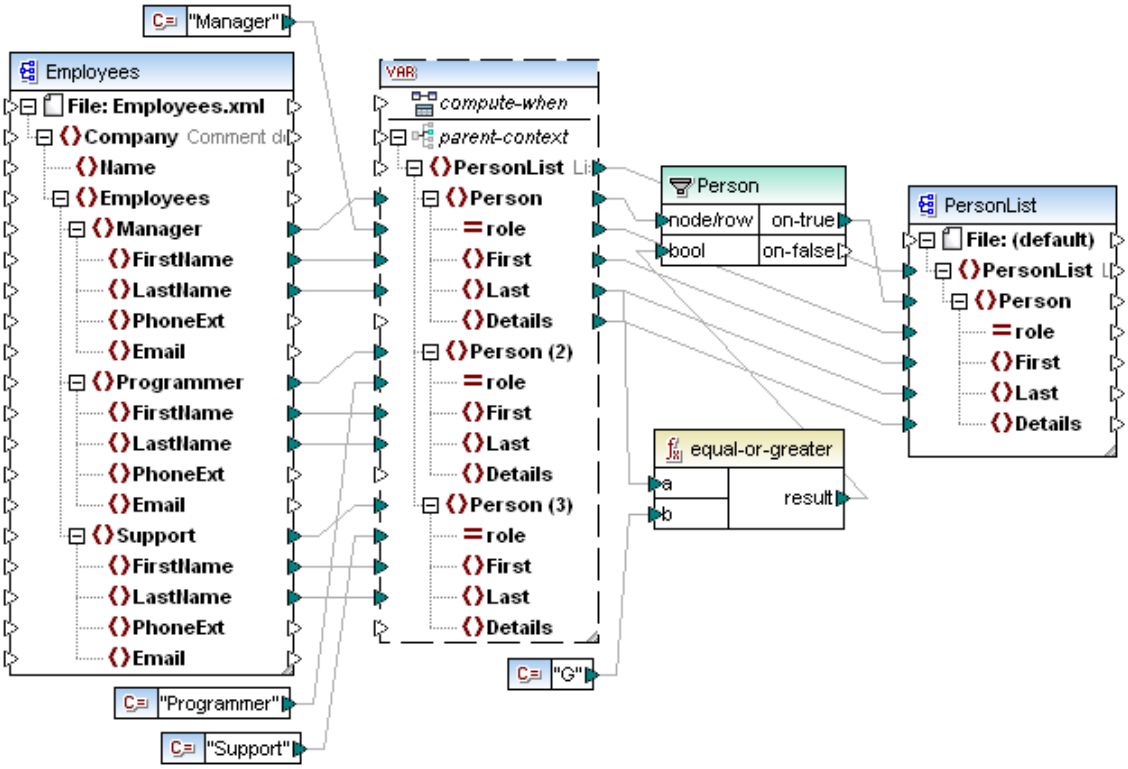
To select the first article of each group, we use the position function and a filter component, which are connected to the variable input.

Applying filters to intermediate sequences:

Nodes in variable components can be duplicated as in any other type of component. This allows you to build sequences from multiple different sources and then further process the sequence.

The screenshot below shows how PersonList.mfd could be modified using an intermediate variable, and how constant components can also act as source items.

© 2015 Altova GmbH

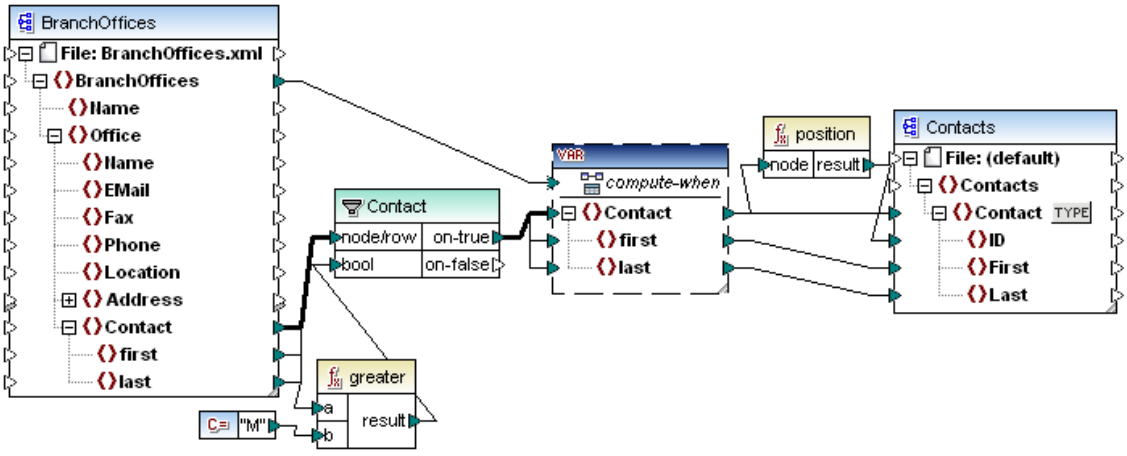


The Person node of the **variable** has been duplicated twice, and a filter has been added to filter those persons whose Last name starts with a letter higher than "G".

Numbering nodes of a filtered sequence

This example is available as **PositionInFilteredSequence.mfd** in ...\MapForceExamples folder and uses the variable to collect the filtered contacts where the last name starts with a letter higher than M.

The contacts are then passed on (from the variable) to the target component, with the position function numbering these contacts sequentially.

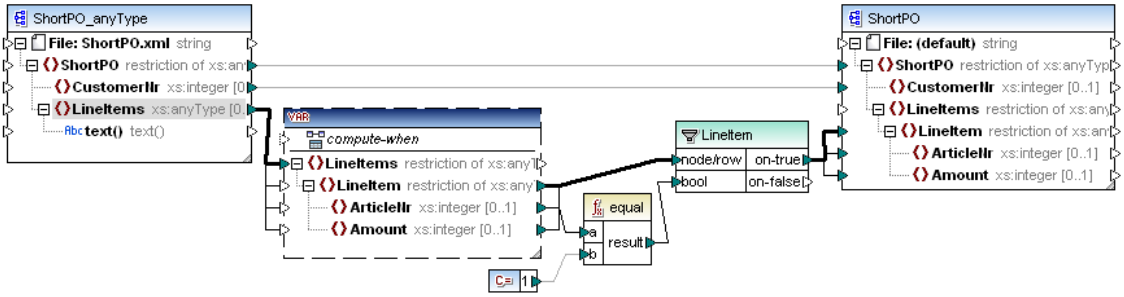


The variable acts like another source component allowing the position function to process the

filtered sequence.

Extract specific data from a source components' anyType node

This example consists of a source component containing anyType elements from which we want to filter specific data.



The intermediate variable is based on a schema that has nodes of the specific type of data that we want to map, i.e. ArticleNr and Amount are both of type integer. That specific data is filtered by ArticleNr. and passed on to the target component.

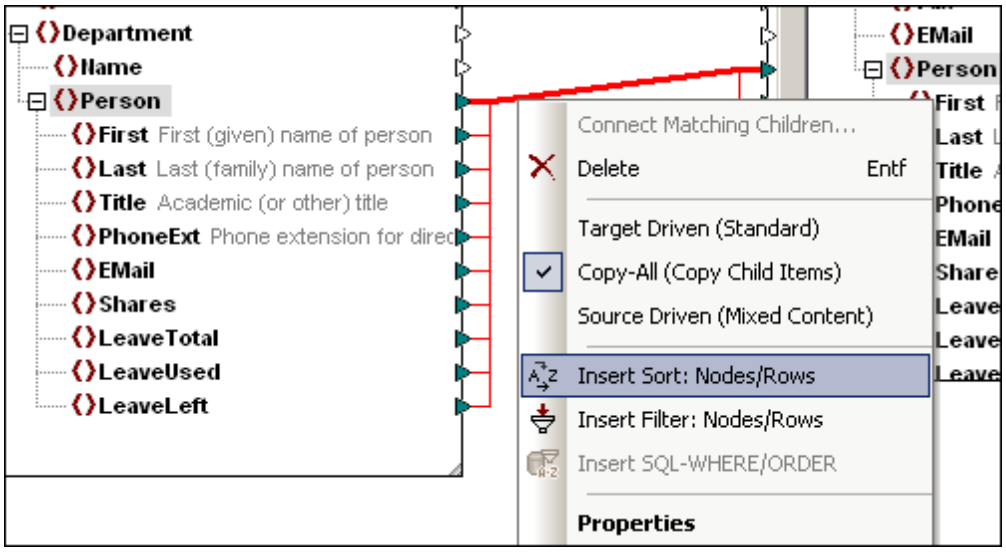
 © 2015 Altova GmbH

5.8 Sorting Data

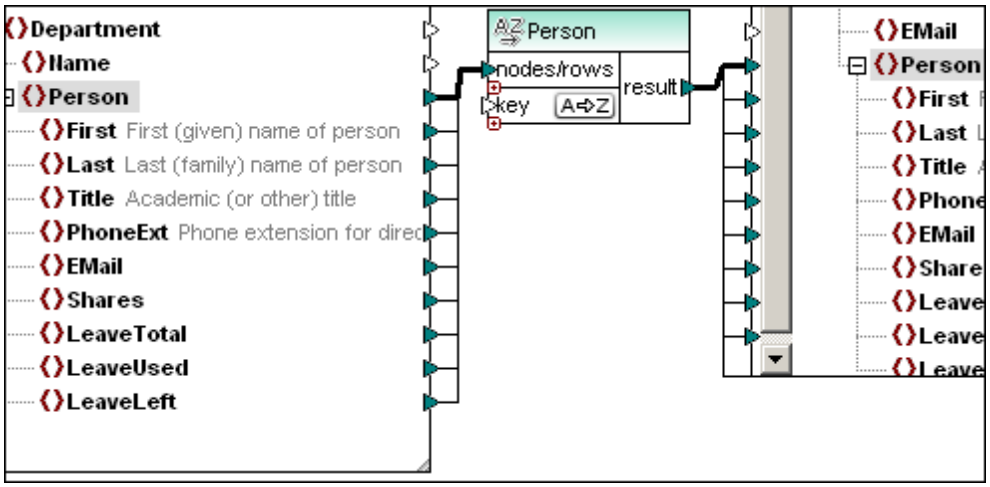
To sort input data based on a specific sort key, use the **Sort** component. The sort component currently supports the targets: XSLT2, XQuery, and the Built-in execution engine.

To insert a Sort component:

1. Right click a connector that exists between the nodes that you want to sort.



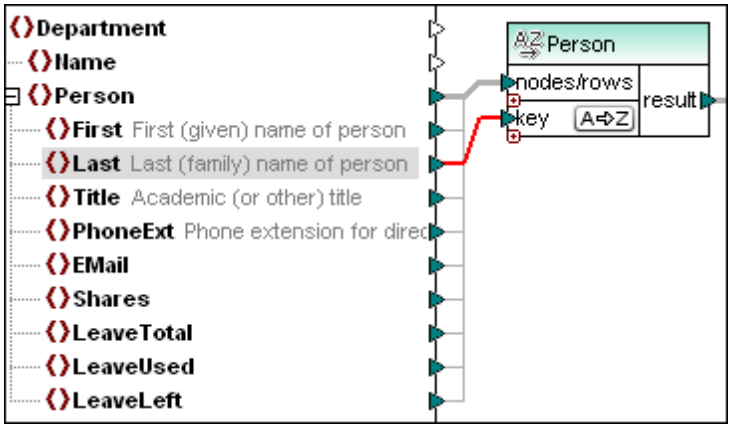
2. Click the **Insert Sort: Nodes/Rows** item from the context menu.



This inserts, and automatically connects, the sort component to the source and target components.

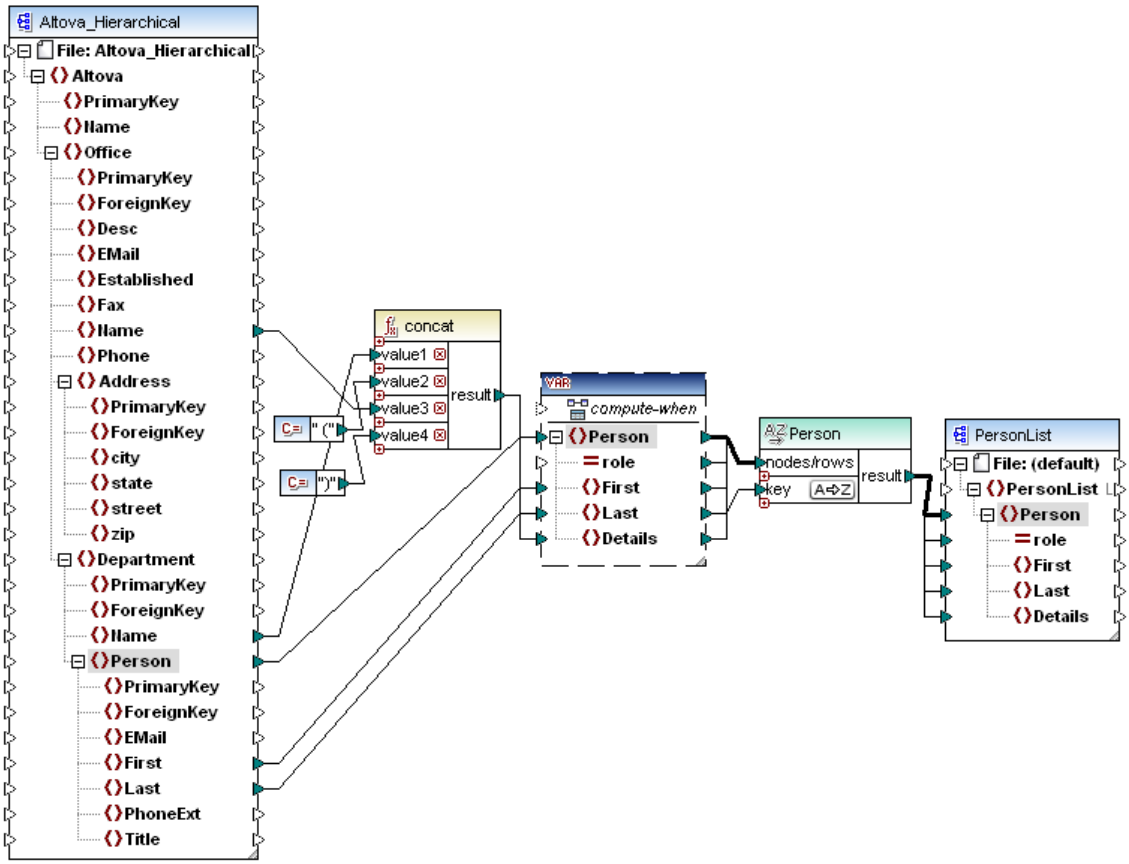
To define which item you want to sort by:

- Connect the item you want to sort by, e.g. **Last**, to the **key** parameter of the sort component, now named "Person".



The Persons wil now be sorted by Last in the output tab.

Example: **Altova_Hierarchical_Sort.mfd** in the MapForceExamples folder.

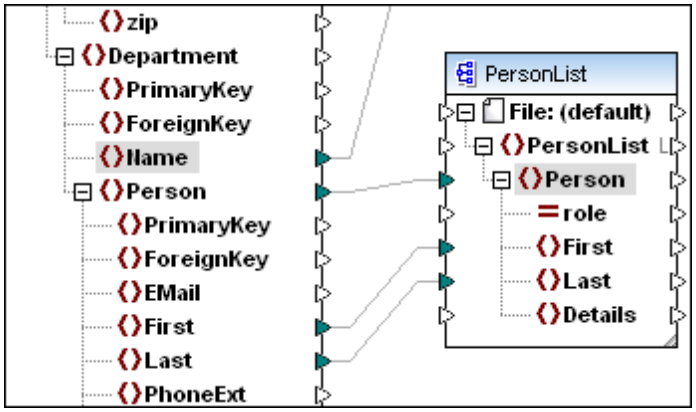


The aim is to have the persons of each of the branch offices alphabetically sorted, and also include detailed information on the office and department names. This example makes use of the Variable component which allows access to otherwise unavailable parent items. In this case parent items of the Person node.

How this mapping was created:

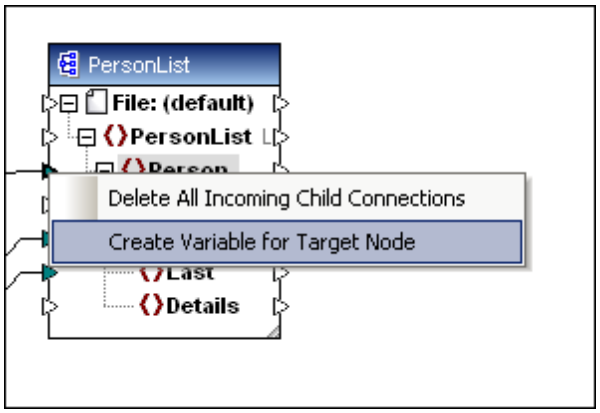
 © 2015 Altova GmbH

The initial stage of the mapping is shown below.

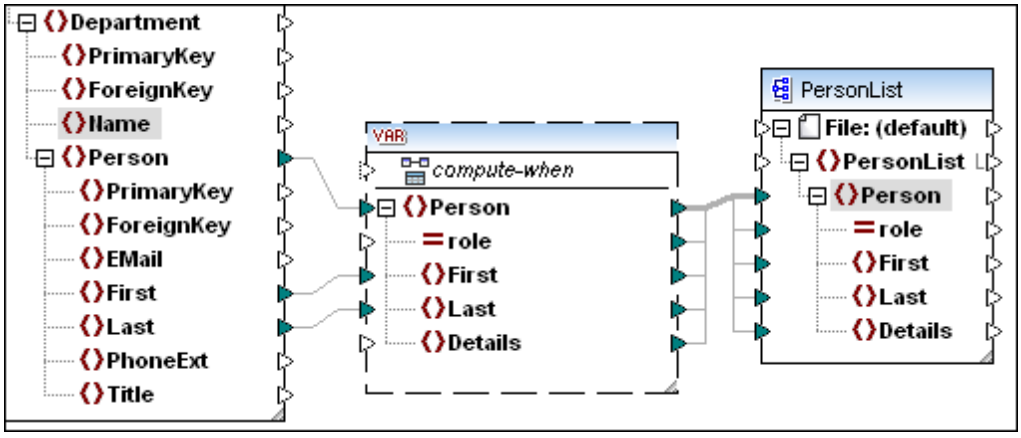


As we want to sort a Person item, the Person items in both components are connected, which also connects the identically named child items.

1. Right click the input icon (exactly on the triangle) next to the Person item of the target component.
2. Select the "Create Variable for Target Node" item in the context menu.

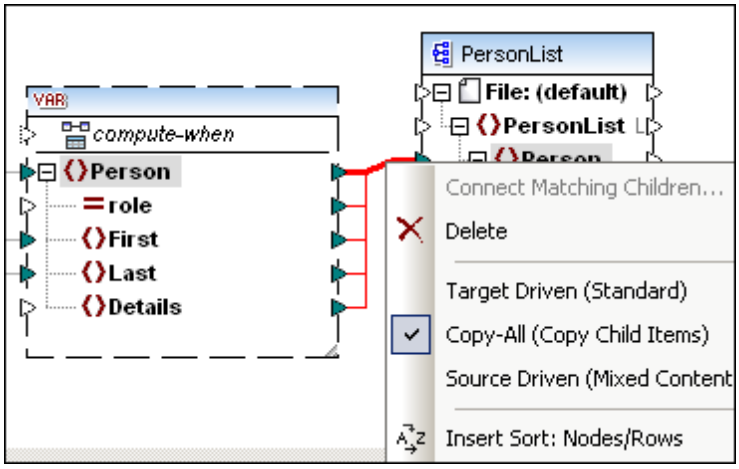


This inserts a complex Intermediate variable between the components and connects the identical items. See Intermediate variables for more specific information.

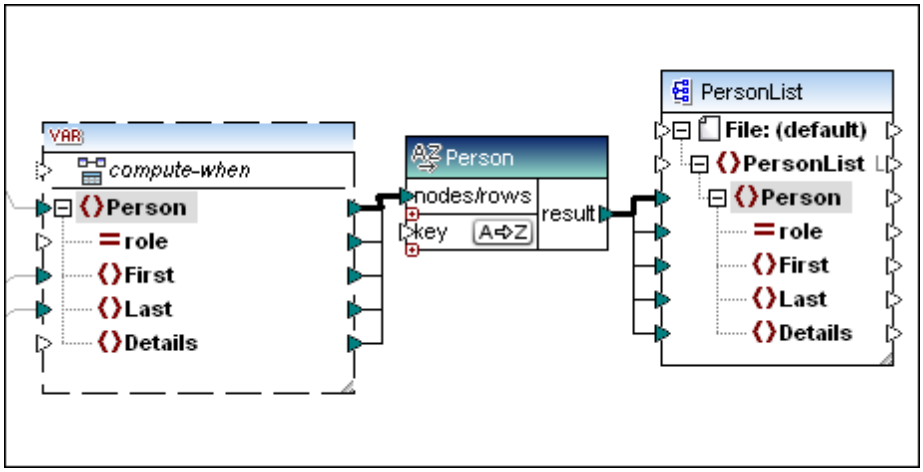


As we want the persons to be sorted by their last names, we now need to insert the sort component.

3. Right click the Copy-All connector and select Insert Sort: Nodes/Rows.

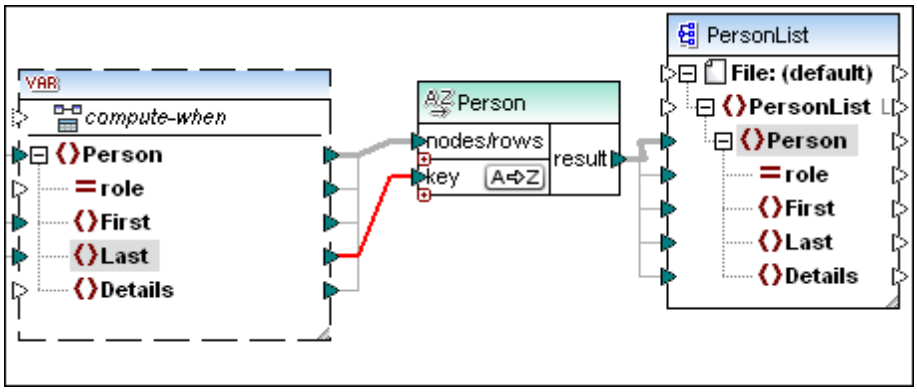


This inserts the sort component and automatically connects all the relevant items.



The only thing left to do is to define the key that we want to sort by.

4. Connect the Last item of the intermediate variable to the **key** parameter of the sort component.

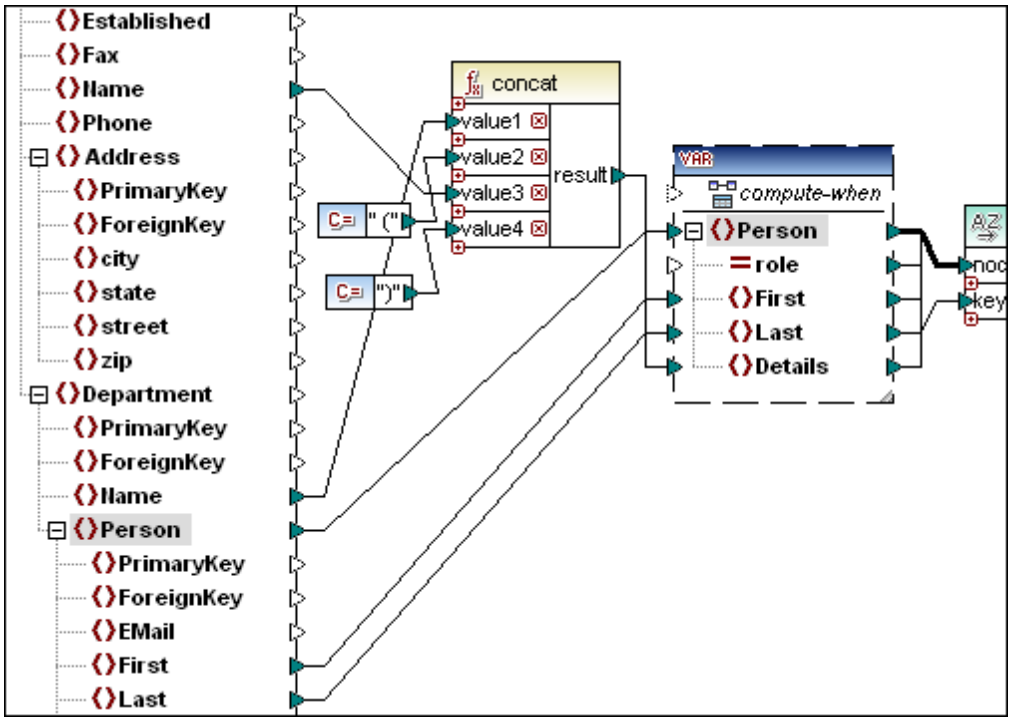


5. Click the Output button to see a preview of the result.
   The persons are now sorted by last name. Click the Mapping button to return to the design window.
6. Using the concat function, and the constant components (to supply the parenthesis)

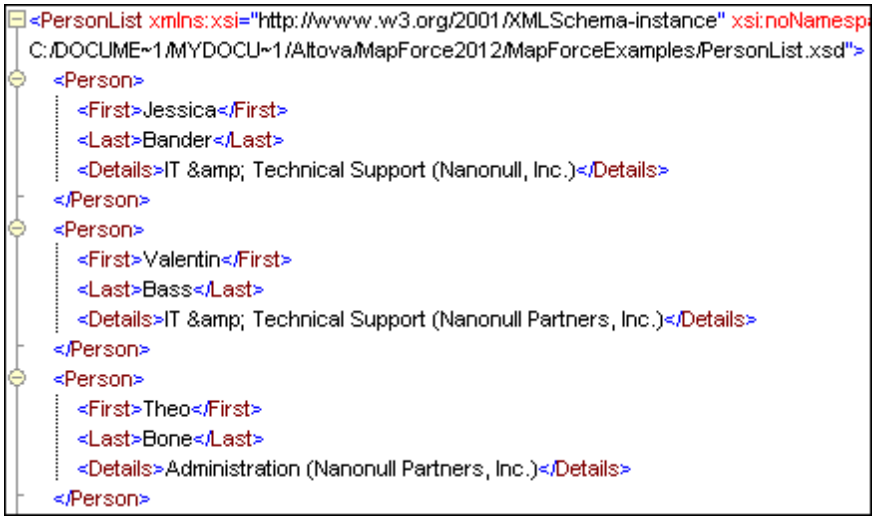
 © 2015 Altova GmbH

connect up the other items as shown below.

7. Connect the result parameter of the concat function, to the Details item of the intermediate variable.



8. Click the Ouput button to see the result.



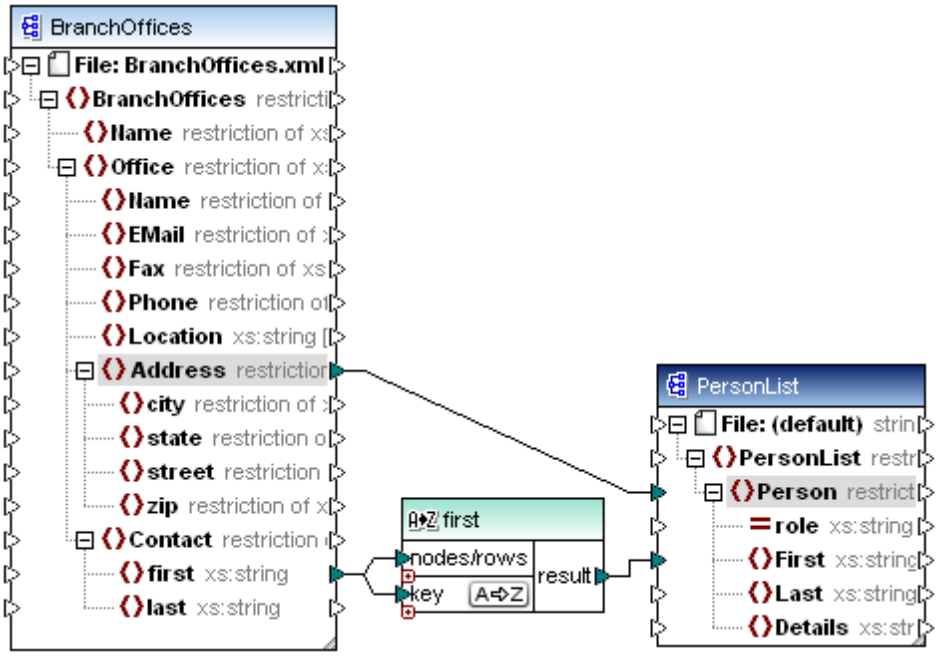
The persons are still sorted by last name but additional info, supplied by the details field, has been added to each person. The correct office and department names are now available to each person, because the intermediate variable makes it possible to access parent data from a child node. This is only possible by using the intermediate variable.

To reverse the sort sequence:

- Click the A⇨Z icon in the Sort component. It changes to Z➔A to show that the sequence has been reversed.

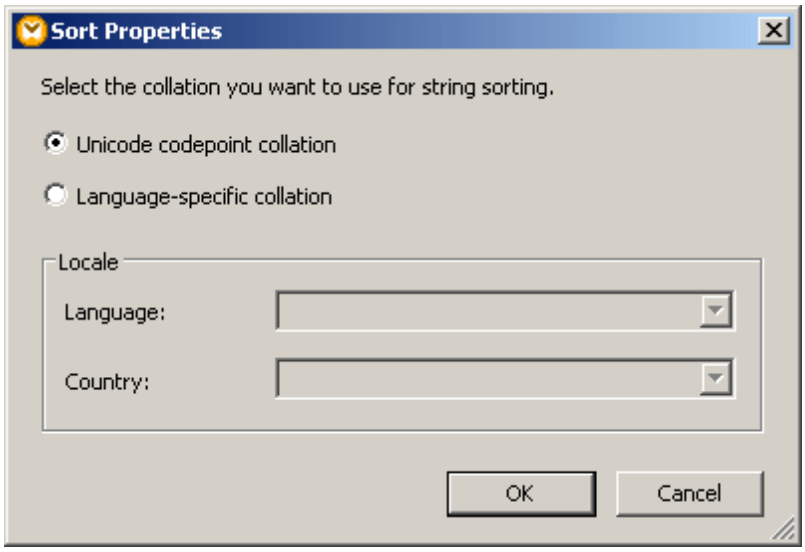
To sort input data consisting of simple type items:

- Connect the simple content item, e.g. first, to **both** the nodes/row and key parameters of the sort component.



To sort strings using language-specifc rules:

Double click the title bar/header of the inserted Sort component to open the Sort Properties dialog box.



Unicode code point collation: This (default) option compares/orders strings based on code point values. Code point values are integers that have been assigned to abstract characters in the Universal Character Set adopted by the Unicode Consortium. This option allows sorting across

 © 2015 Altova GmbH

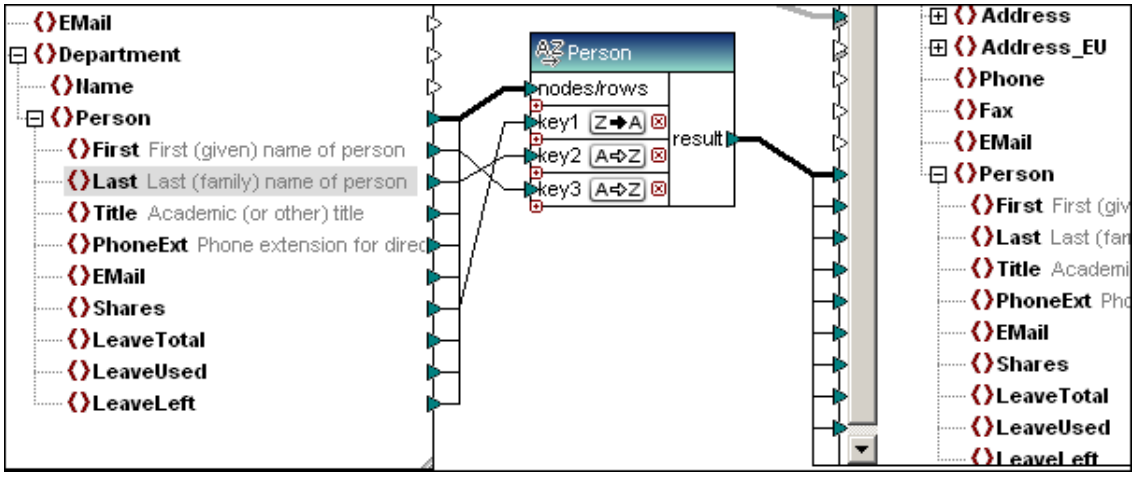
many languages and scripts.

Language-specific collation: This option allows you to define the specific language and country variant you want to sort by. This option is supported when using the BUILTIN execution engine, and for XSLT support depends on the specific engine used to execute the code.

To sort by multiple keys:

The sort component allows you to define multiple keys.

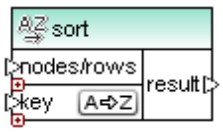
- Clicking the "+" icon adds a new key to the sort component, i.e. key2
- Clicking the "x" icon deletes a that specific key.
- Dropping a connector onto a + icon automatically inserts/adds the parameter and connects to it.



Note that key1 has a higher sort priority than key2, and key2 higher than key3.

To insert a Sort component conventionally:

- Click the Sort icon in the icon bar to insert the component
  This inserts the sort component in its "unconnected" form where "sort" is visible in the title bar of the component.



As soon as a connection is made to the source component, the title bar name changes to that of the item connected to the nodes/rows item.

5.9 Filtering Data

This section will deal with methods enabling you to optimize data access and generally speed up the mapping process.

In general, use as few filter components as possible, and:

1. Avoid concatenating filter-components
2. Connect the "on-true/on-false" parameters, to parent items if possible, instead of child items directly
3. Connect the "on-false" parameter to map the complement node set, delivered by the on-true parameter
4. Don't use filters to map to child data, if the parent item is mapped
5. Use the "Priority context" to prioritize execution of unrelated items

Avoid concatenating filter components
Every filter-component leads to a loop through the source data, thus accessing the source **n** times. When you concatenate two filters, it loops **n*n** times.

Solution:
Use "**logical-and**" components to combine the boolean expressions of two filter-components. The result is a single filter component looping only **n**-times.

Connect the "on-true/on-false" parameter of the filter component, to target parent items
Filter components work best when they are connected to parent items containing child items, instead of individual items directly.

The filter **boolean** expression is therefore evaluated against the parent, **before** looping through the child elements. Using filters mapped from a database table will generate:

- "SELECT * FROM table WHERE <expression>" if the **parent** item is mapped, or
- "SELECT * FROM table", and then evaluate for each row, if child items are mapped

Please note:
when connecting a filter from a source parent item, it is also necessary to connect the on-true/on-false parameter to the parent target element. If this cannot be done, then do not apply this rule.

Connect the "on-false" parameter to map the complement node set
Connecting this parameter allows you quick access to the complement node set defined by the current mapping. The same tips apply when using this parameter, connect to parent items etc.

Don't use filters to map to child data, if the parent item is mapped
Using a filter to map data from a source parent to a target parent, automatically applies the **same filter** to **every child** item of the particular parent.

Filter components do not have to be used to supply filtered data to child items, if the parent item can be mapped! You can therefore map child data directly.

Use priority-context to prioritize execution when mapping unrelated items
Mappings are always executed top-down; if you loop/search through two tables then each loop is processed consecutively. When mapping unrelated elements, without setting the priority context,

 © 2015 Altova GmbH

MapForce does not know which loop needs to be executed first, it therefore automatically selects the first table, or data source.

Solution:
Decide which table, or source data is to be looped/searched first, and then set the priority context on the connector to that table. Please see Priority Context node/item for a more concrete example.

To define a priority context:

- Right click an input icon and select "Priority Context" from the pop-up menu.
  If the option is not available, mapping the remaining input icons of that component will make it accessible.

Filters and source-driven / mixed content mapping

Source-driven mappings only work with direct connections between source and target components. Connections that exist below a source-driven connection, are not taken as source-driven and the items will be handled in target component item/node order.

A single filter where both outputs are connected to same/separate targets, acts as if there were two separate filter components, one having a negated condition.

If an exception component is connected to one of the filter outputs, the exception condition is checked when the mappings to the the other filter output are executed.

5.10 Using Value-Maps

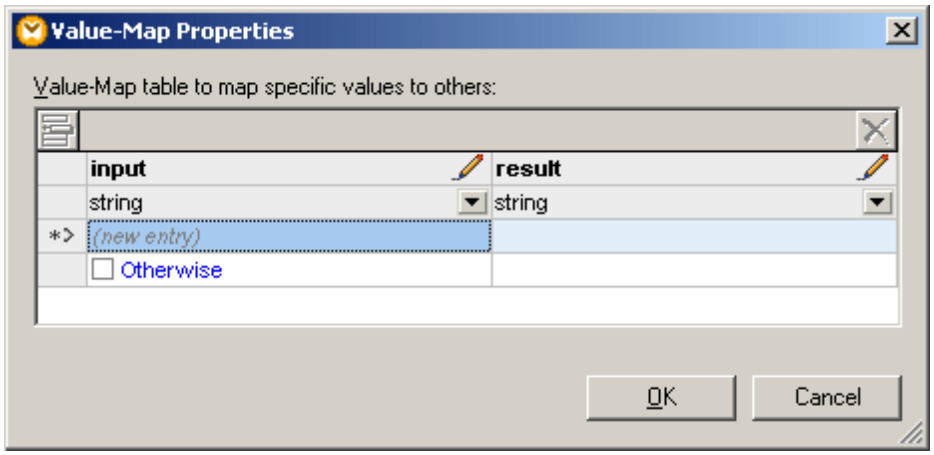
The Value-Map component allows you to transform an input value to a different output value using a lookup table. This is useful for converting different enumeration types. The component only has one input and output item.

Note: if you want to retrieve/filter data based on specific criteria, please use the **Filter** component, see Filtering.

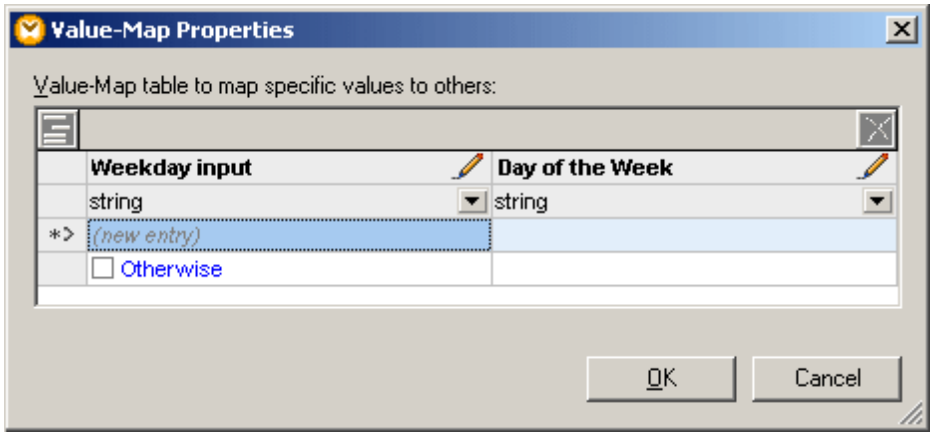
To use a Value-Map component:

1. Select the menu option **Insert | Value-Map**, or click the Value-Map icon in the icon bar.

2. Double click the Value-Map component to open the value map table.



3. Click into the column headers and enter **Weekday input** in the first, and **Day of the Week** in the second.

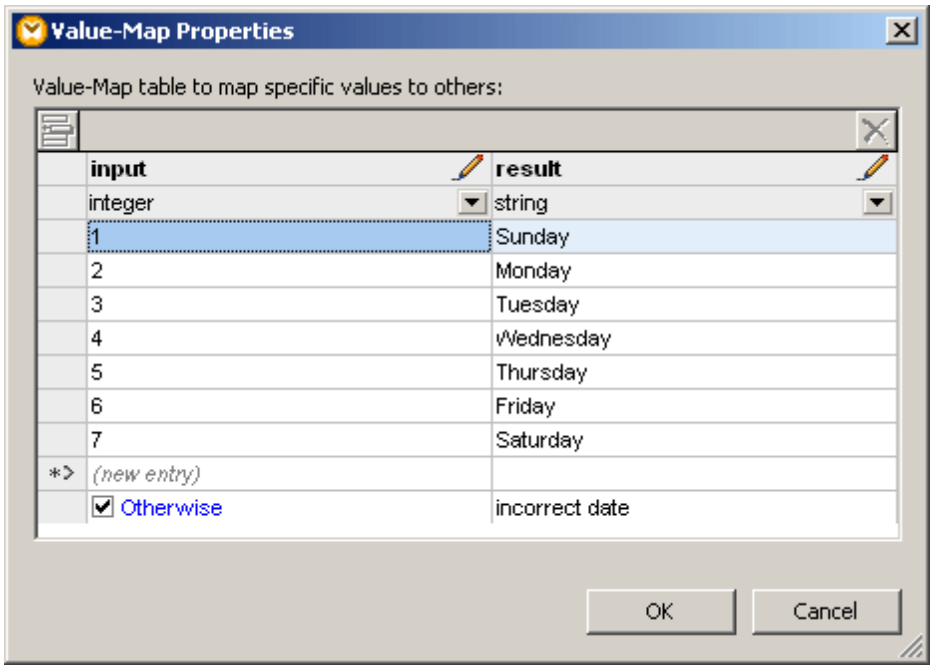


4. Enter the input value that you want to transform, in the **Weekday input** column.
5. Enter the output value you want to transform that value to, in the **Day of the week**

 © 2015 Altova GmbH

column.

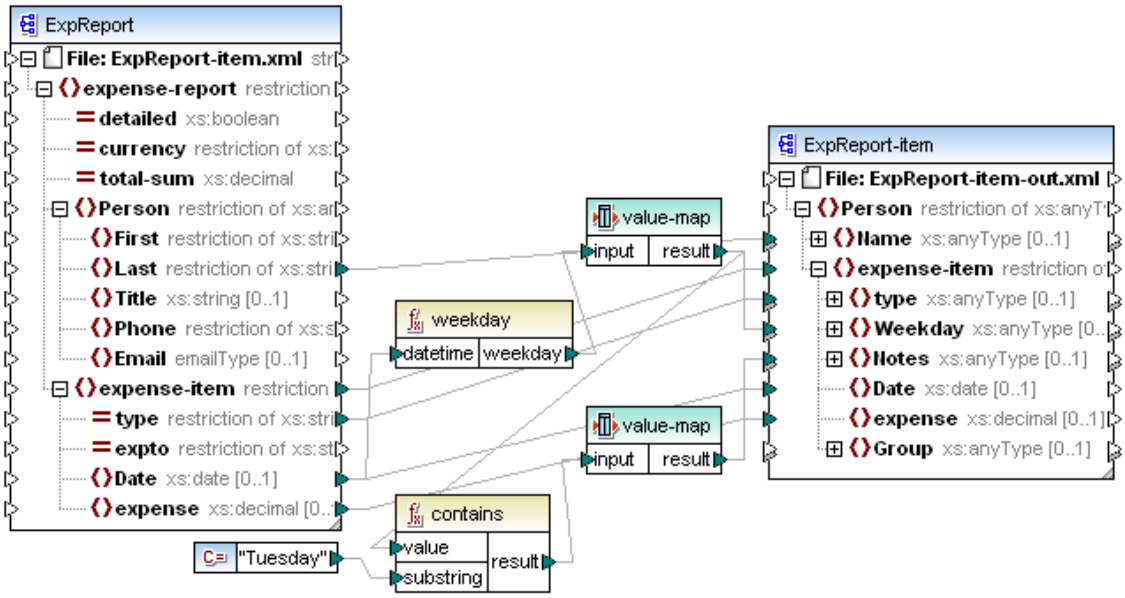
6. Simply type in the ***(new entry)*** input field to enter a new value pair.
7. Click the **datatype** combo box, below the column header to select the input and output datatypes, e.g. integer and string.



Note: activate the **Otherwise** check box, and enter the value, to define an alternative output value if the supplied values are not available on input. To pass through source data without changing it please see Passing data through a Value-Map unchanged.

8. You can click the edit icons in the header rows to change the column names, which are also displayed in the mapping. This will make it easier to identify the purpose of the component in the mapping.

The **Expense-valmap.mfd** file in the ...\MapForceExamples\Tutorial\ folder is a sample mapping that shows how the Value-Map can be used.



What this mapping does:
Extracts the day of the week from the Date item in the data source, converts the numerical value into text, and places it in the Weekday item of the target component i.e. Sunday, Monday etc.

- The **weekday** function extracts the weekday number from the **Date** item in the ExpReport source file. The result of this function are integers ranging from 1 to 7.
- The Value-Map component transforms the integers into weekdays, i.e. Sunday, Monday, etc. as shown in the graphic at the top of this section.
- If the output contains "Tuesday", then the corresponding output "Prepare Financial Reports" is mapped to the Notes item in the target component.
- Clicking the Output tab displays the target XML file with the transformed data.

```
3   <Name>Landis</Name>
4   <expense-item>
5     <type>Meal</type>
6     <Weekday>Tuesday</Weekday>
7     <Notes>-- Prepare financial reports -- !</Notes>
8     <Date>2003-01-01</Date>
9     <expense>122.11</expense>
10  </expense-item>
11  <expense-item>
12    <type>Lodging</type>
13    <Weekday>Monday</Weekday>
14    <Notes> --</Notes>
15    <Date>2003-01-14</Date>
16    <expense>122.12</expense>
17  </expense-item>
```

Note:
Placing the mouse cursor over the value map component opens a popup containing the currently defined values.

The output from various types of logical, or string functions, can only be a boolean "**true**"

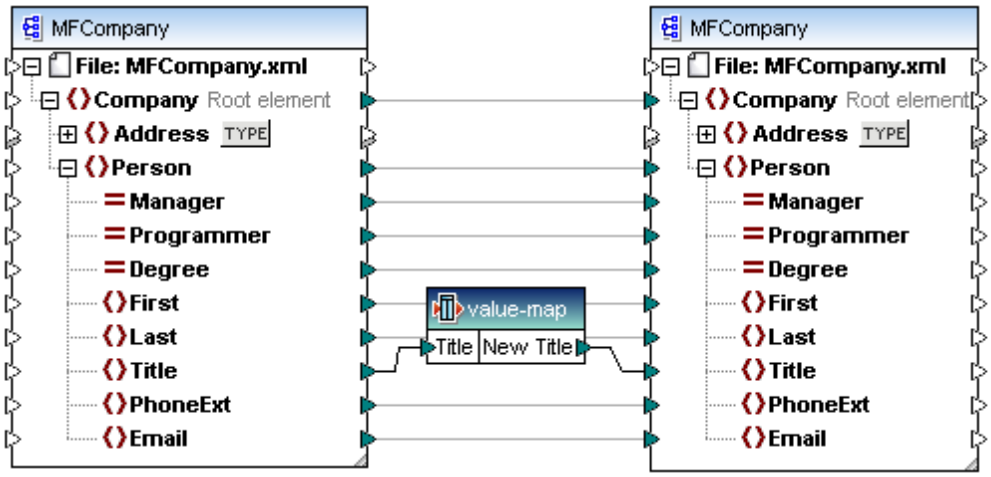
 © 2015 Altova GmbH

or "**false**" value. The value you want to test for, must thus be entered into the **input** field of the value map table e.g. "true".

5.10.1 Passing data through a Value-Map unchanged

This section describes a mapping situation where some specific node data have to be transformed, while the rest of the node data have to be passed on to the target node unchanged.

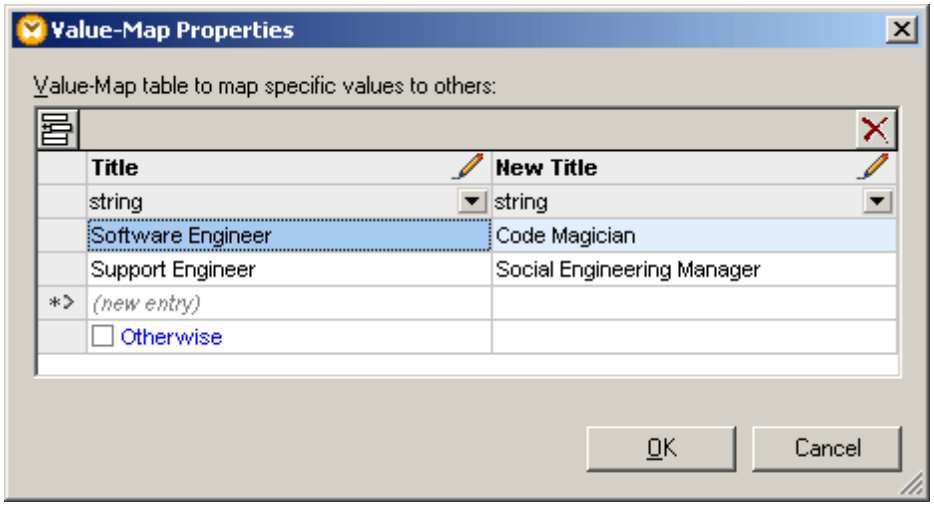
An example of this would be a company that changes some of the titles in a subsidiary. In this case it might change two title designations and want to keep the rest as they currently are.



The obvious mapping would be the one shown above, which uses the value-map component to transform the specific titles.
Clicking the Output tab shows us the result of the mapping:

```
33  <Person>
34    <First>Fred</First>
35    <Last>Landis</Last>
36    <PhoneExt>951</PhoneExt>
37    <Email>f.landis@nanonull.com</Email>
38  </Person>
39  <Person>
40    <First>Michelle</First>
41    <Last>Butler</Last>
42    <Title>Code Magician</Title>
43    <PhoneExt>654</PhoneExt>
44    <Email>m.landis@nanonull.com</Email>
45  </Person>
```

For those persons who are neither of the two types shown in the value-map component, the Title element is deleted in the output file.

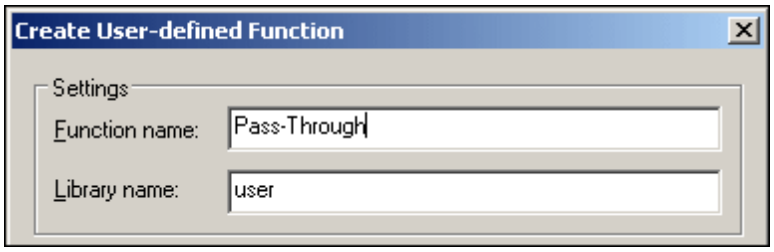


Possible alternative:
Clicking the **Otherwise** check box and entering a substitute term, does make the Title node reappear in the output file, but it now contains the same **New Title** for all other persons of the company.

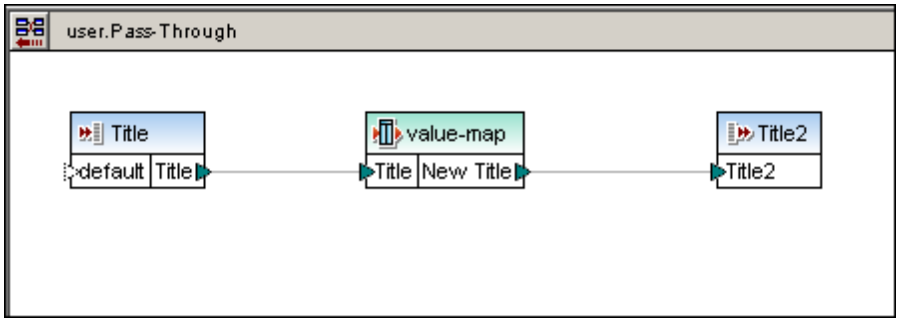
Solution:

Create a user-defined function containing the value-map component, and use the **substitute-missing** function to supply the original data for the empty nodes.

1. Click the value-map component and select **Function | Create user-defined function from Selection**.

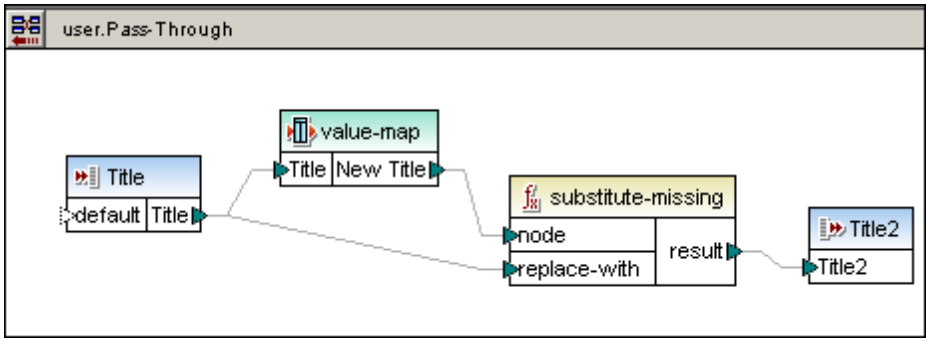


2. Enter a name for the function e.g. Pass-Through and click OK.



3. Insert a **substitute-missing** function from the **core | node function** section of the Libraries pane, and create the connections as shown in the screen shot below.

 © 2015 Altova GmbH



4. Click the Ouput tab to see the result:

Result of the mapping:

- The two Title designations in the value-map component are transformed to New Title.
- All other Title nodes of the source file, retain their original Title data in the target file.

```
38  <Person>
39    <First>Fred</First>
40    <Last>Landis</Last>
41    <Title>Program Manager</Title>
42    <PhoneExt>951</PhoneExt>
43    <Email>f.landis@nanonull.com</Email>
44  </Person>
45  <Person>
46    <First>Michelle</First>
47    <Last>Butler</Last>
48    <Title>Code Magician</Title>
49    <PhoneExt>654</PhoneExt>
50    <Email>m.landis@nanonull.com</Email>
51  </Person>
```

Why is this happening:
The value-map component evaluates the input data.

- If the incoming data **matches one** of the entries in the first column, the data is transformed and passed on to the node parameter of substitute-missing, and then on to Title2.

- If the incoming data does not match **any entry** in the left column, then nothing is passed on from value-map to the node parameter i.e. this is an **empty node**.

  When this occurs the substitute-missing function retrieves the orignal node and data from the Title node, and passes it on through the **replace-with** parameter, and then on to Title2.

5.10.2 Value-Map component properties

Actions:

Click the insert icon to **insert** a new row before the currently active one.

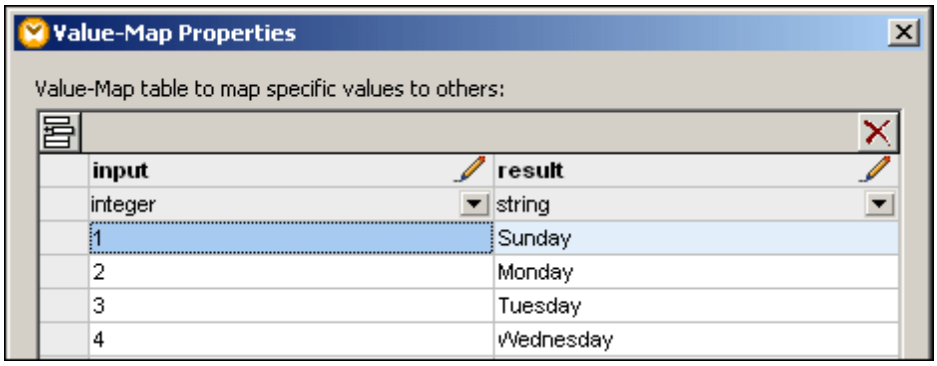
Click the delete icon to **delete** the currently active row.

Click the edit icon to **edit** the column header.

You can also reorder lines by dragging them.

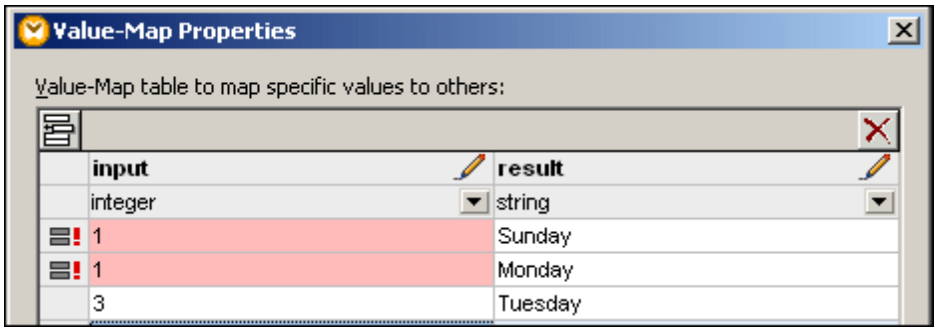
Changing the column header:

Double clicking the column header, or clicking the pencil icon, allows you to edit the column name and change it to something more meaningful. This will make it easier to identify the purpose of the component, as the column names are also displayed in the mapping.



Using unique Input values:

The values entered into the input column must be unique. If you enter two identical values, both are automatically highlighted for you to enable you to correct one of them.



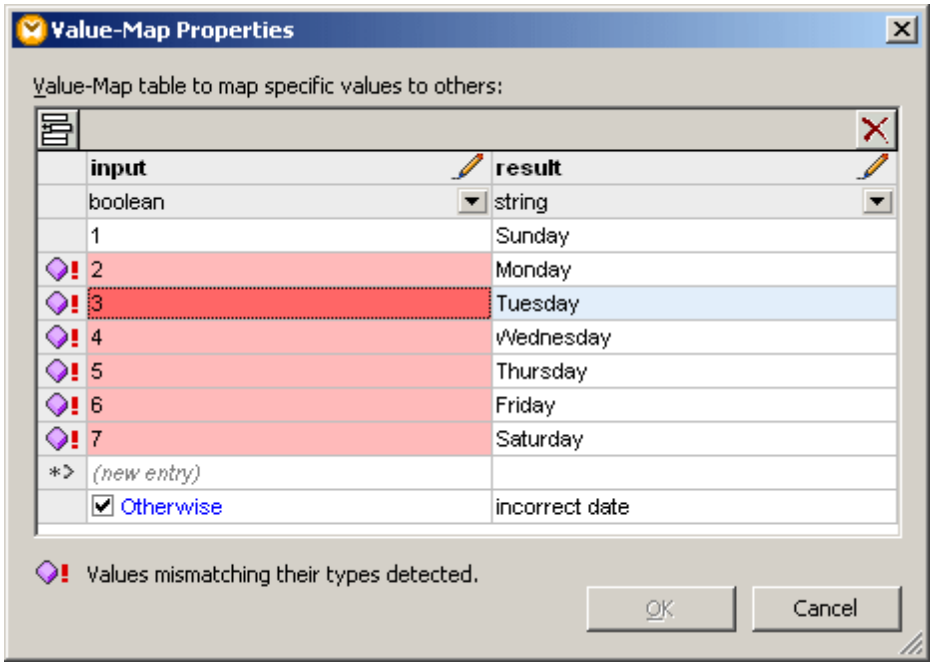
Having corrected one of the values, the OK button is again enabled.

Input and output datatypes

The input and result datatypes are automatically checked when a selection is made using the combo box. If a mismatch occurs, then the respective fields are highlighted and the OK button is disabled. Change the datatype to one that is supported.

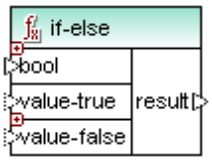
In the screenshot below a boolean and string have been selected.

 © 2015 Altova GmbH


Value-Map Properties
Value-Map table to map specific values to others:
input
result
boolean
string
1
Sunday
2
Monday
3
Tuesday
4
Wednesday
5
Thursday
6
Friday
7
Saturday
(new entry)
Otherwise
incorrect date
Values mismatching their types detected.
OK
Cancel

5.11 Using If-Else Conditions

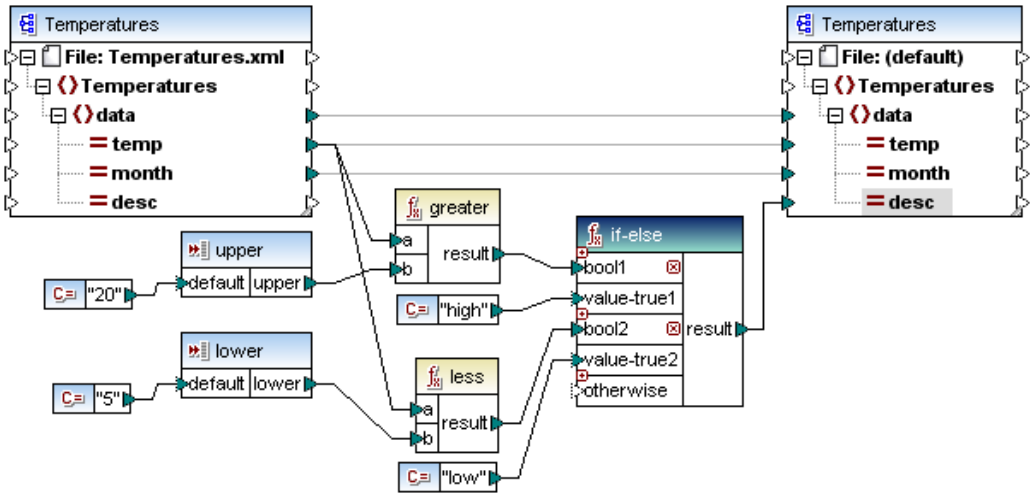
An **If-Else Condition** is a special component type which allows you to pass on different sets of data depending on the outcome of a preset condition. The component header displays the text **if-else**.



- The first input parameter is a **bool**, which contains the data you are checking against.
- The **value-true** input parameter supplies the data to be passed on, as a result, if the condition is true.
- The **value-false** supplies the data to be passed on if the condition is false.
- The **result** parameter outputs the data supplied by the value-true/false input parameters.

The IF-Else function is **extendable**. This means that you can check for multiple conditions and use the **otherwise** parameter to output the Else condition/value.

Clicking the "plus" icon inserts or appends a new if-else **pair**, i.e. boolX and value-trueX, while clicking the "x" deletes the parameter pair.

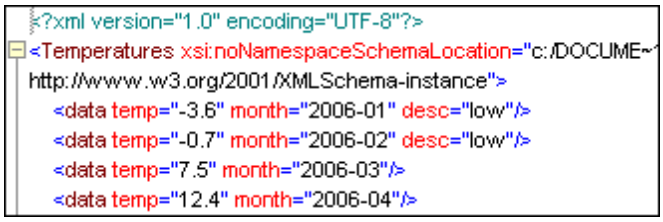


In the example above, the temperature data is analyzed:

- If temp is **greater** than 20, then true is passed on to **bool1** and the result is "**high**" from **value-true1.**
- Else, If temp is **less** than 5, then true is passed on to **bool2** and the result is "**low**" from **value-true2.**
- Otherwise, nothing (an empty sequence) is the result of the component, since there is no connection to the "otherwise" input.

Result of the mapping:

 © 2015 Altova GmbH


<?xml version="1.0" encoding="UTF-8"?>
<Temperatures xsi:noNamespaceSchemaLocation="c:/DOCUME~
http://www.w3.org/2001/XMLSchema-instance">
<data temp="-3.6" month="2006-01" desc="low"/>
<data temp="-0.7" month="2006-02" desc="low"/>
<data temp="7.5" month="2006-03"/>
<data temp="12.4" month="2006-04"/>

5.12 Mapping Rules and Strategies

MapForce generally maps data in an intuitive way, but you may come across situations where the resulting output seems to have too many, or too few items. This topic is intended to help you avoid such mapping problems.

General rule

Generally, every connection between a source and target item means: for each source item, create one target item. If the source node contains simple content (for example, string or integer) and the target node accepts simple content, then MapForce copies the content to the target node and, if necessary, converts the data type.

This generally holds true for all connections, with the following exceptions:

- A target XML root element is always created once and only once (see also Mapping to the Root Element). If you connect a sequence to it, only the contents of the element will be repeated, but not the root element itself, and the result might not be schema-valid. If attributes of the root element are also connected, the XML serialization will fail at runtime, so you should avoid connecting a sequence to the root element. If what you want to achieve is creating multiple output files, connect the sequence to the "File" node instead, via some function that generates file names.
- Some nodes accept a single value, not a sequence (for example, XML attributes, , and output components in user-defined functions).

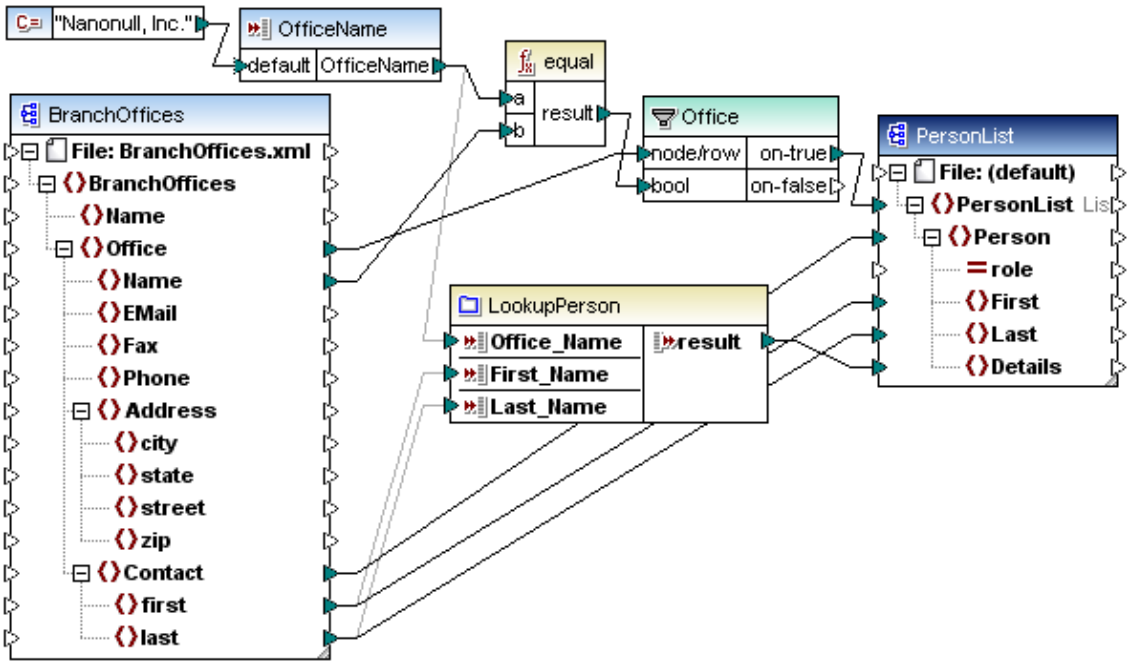
The "context" and "current" items

MapForce displays the structure of a schema file as a hierarchy of mappable items in the component. Each of these nodes may have many instances (or none) in the instance file or database.

Example: If you look at the source component in **PersonListByBranchOffice.mfd**, there is only a single node **first** (under **Contact**). In the **BranchOffices.xml** instance file, there are multiple **first** nodes and **Contact** nodes having different content, under different **Office** parent nodes.

It depends on the current **context** (of the **target** node) which source nodes are actually selected and have their data copied, via the connector, to the target component/item.

© 2015 Altova GmbH



PersonListByBranchOffice.mfd

This context is defined by the **current target node** and the connections to its ancestors:

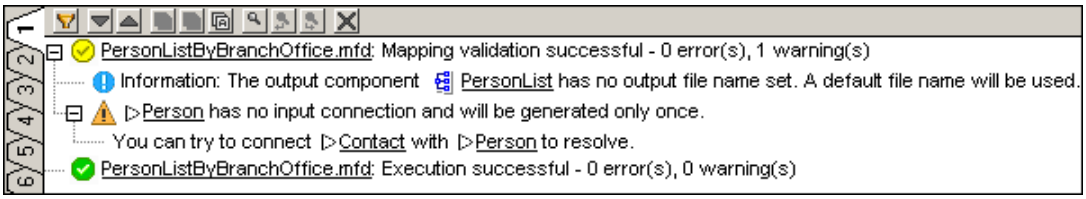
- Initially the context contains only the source components, but no specific nodes. When evaluating the mapping, MapForce processes the **target root** node first (PersonList), then works down the hierarchy.
- The connector to the **target** node is traced back to all source items directly or indirectly connected to it, even via functions that might exist between the two components. The source items and functions results are added to the context for this node.
- For each new target node a new context is established, that initially contains all items of the parent node's context. Target sibling nodes are thus independent of each other, but have access to all source data of their parent nodes.

Applied to the example mapping above (**PersonListByBranchOffice.mfd**):

- The connection from **Office** through the filter (Office) to **PersonList** defines a **single** office as the context for the whole target document (because PersonList is the root element of the target component). The office name is supplied by the input component, which has a default containing "Nanonull, Inc."
- All connections/data to the **descendants** of the root element PersonList, are automatically affected by the filter condition, because the selected single office is in the context.
- The connection from **Contact** to **Person** creates one target Person **per** Contact item of the source XML (general rule). For each Person one specific Contact is added to the context, from which the children of Person will be created.
- The connector from **first** to **First** selects the first name of the current Contact and writes it to the target item First.

Leaving out the connector from **Contact** to **Person** would create only **one** Person with multiple

First, Last, and Detail nodes, which is not what we want here. In such situations, MapForce issues a warning and a suggestion to fix the problem: "You can try to connect Contact with Person to resolve":


PersonListByBranchOffice.mfd: Mapping validation successful - 0 error(s), 1 warning(s)
Information: The output component PersonList has no output file name set. A default file name will be used.
Person has no input connection and will be generated only once.
You can try to connect Contact with Person to resolve.
PersonListByBranchOffice.mfd: Execution successful - 0 error(s), 0 warning(s)

Sequences

MapForce displays the structure of a schema file as a hierarchy of mappable items in the component.

Depending on the (target) context, each **mappable item** of a **source** component can represent:

- a **single instance** node of the assigned input file
- a **sequence** of 0 to **multiple instance** nodes of the input file

If a sequence is connected to a **target** node, a loop is created to create as many target nodes as there are source nodes.

If a **filter** is placed between the sequence and target node, the bool condition is checked for each input node i.e. each item in the sequence. More exactly, a check is made to see if there is at least one bool in each sequence that evaluates to true. The priority context setting can influence the order of evaluation, see below.

As noted above, filter conditions automatically apply to all descendant nodes.

Note: If the source schema specifies that a specific node occurs exactly once, MapForce may remove the loop and take the first item only, which it knows must exist. This optimization can be disabled in the source Component Settings dialog box (check box "Enable input processing optimizations based on min/maxOccurs").

Function inputs (of normal, non-sequence functions) work similar to target nodes: If a sequence is connected to such an input, a loop is created around the function call, so it will produce as **many results** as there are items in the sequence.

If a sequence is connected to **more than one** such function input, MapForce creates nested loops which will process the **Cartesian product** of all inputs. Usually this is not desired, so only one single sequence with multiple items should be connected to a function (and all other parameters bound to singular current items from parents or other components).

Note: If an empty sequence is connected to such a function (e.g. concat), you will get an **empty sequence** as result, which will produce no output nodes at all. If there is no result in your target output because there is no input data, you can use the “substitute-missing” function to insert a substitute value.

Functions with **sequence inputs** are the only functions that can produce a result if the input sequence is **empty**:

- **`exists`**, **`not-exists`** and **`substitute-missing`** (also, **`is-not-null`**, **`is-null`** and **`substitute-null`**, which are aliases for the first three)

 © 2015 Altova GmbH

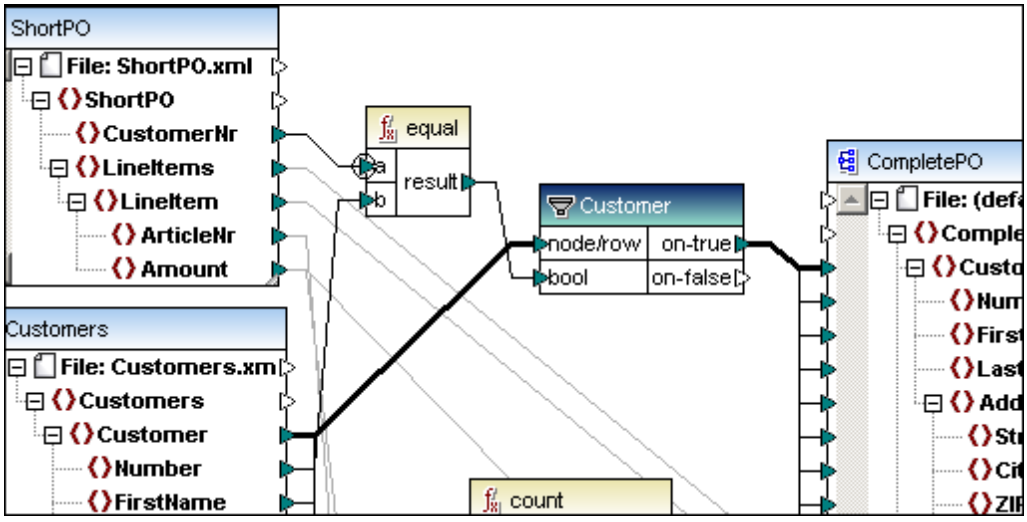
- aggregate functions (**sum**, **count**, etc.)
- regular user-defined functions that accept sequences (i.e. non-inlined functions)

The sequence input to such functions is always evaluated independently of the current target node in the context of its ancestors. This also means that any filter components connected to such functions, do not affect any other connections.

Priority context

Usually, function parameters are evaluated from top to bottom, but its is possible to define one parameter to be evaluated before all others, using the **priority context** setting.

In functions connected to the bool input of **filter** conditions, the priority context affects not only the comparison function itself but also the evaluation of the filter, so it is possible to join together two source sequences (see CompletePO.mfd, CustomerNo and Number).

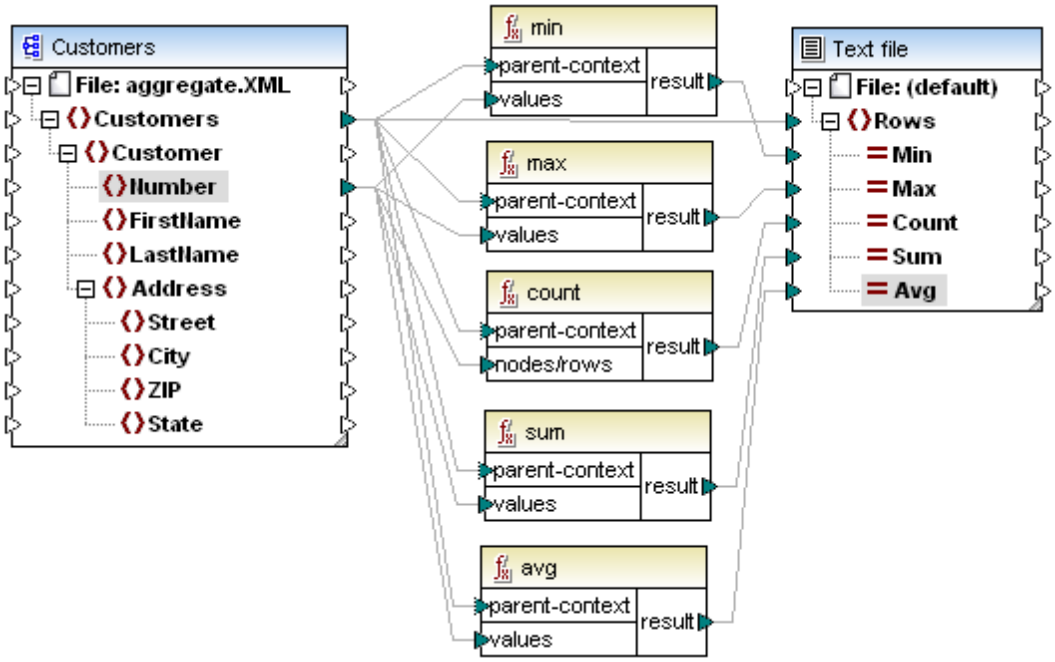


In this example, the priority context forces ShortPO/CustomerNr to be evaluated before iterating and filtering the Customer nodes from the Customers component. See example Priority Context node/item

Overriding the context

Some aggregate functions have an optional “parent-context” input.

If this input is not connected, it has no effect and the function is evaluated in the normal context for sequence inputs (that is, in the context of the target node's parent).



If the parent-context input is connected to a source node, the function is evaluated **for each** "parent-context" node and will produce a separate result for each occurrence.

Bringing multiple nodes of the same source component into the context
This is required in some special cases and can be done with Intermediate variables.

5.12.1 Changing the Processing Order of Mapping Components

MapForce supports mappings that have several target components. Each of the target components has a preview button allowing you to preview the mapping result for that specific component.

If the mapping is executed from the command line or from generated code, then, regardless of the currently active preview, the full mapping is executed and the output for each target component is generated.

The order in which the target components are processed can be directly influenced by changing the position of target components in the mapping window. The **position** of a component is defined as its top left corner.

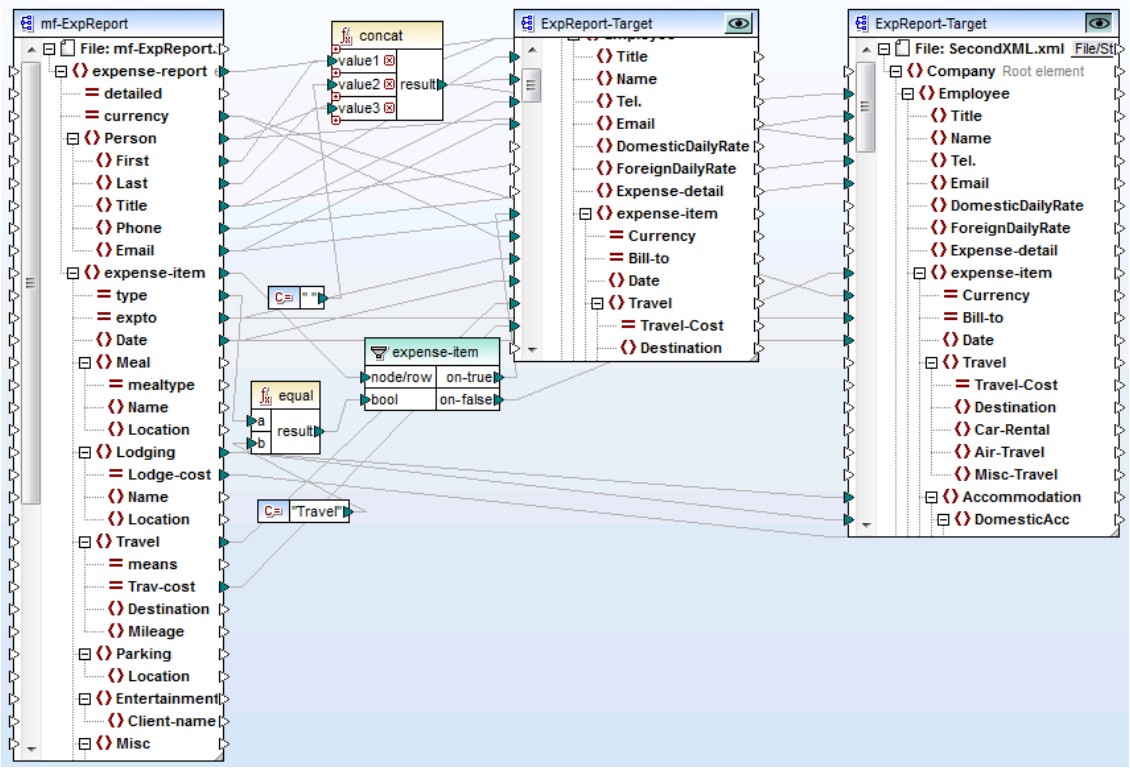
Target components are processed according to their Y-X position on screen, from top to bottom and left to right.

- If two components have the same vertical position, then the leftmost takes precedence.
- If two component have the same horizontal position, then the highest takes precedence.
- In the unlikely event that components have the exact same position, then an unique internal component ID is automatically used, which guarantees a well-defined order but which cannot be changed.

The screenshot below shows the tutorial sample **Tut-ExpReport-multi.mfd** available in the

 © 2015 Altova GmbH

<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial folder. Both target components (ExpReport-Target) have the same **vertical** position, and the preview button is active on the right hand target component.



Tut-ExpReport-multi.mfd (MapForce Enterprise Edition)

Having selected XSLT2 and generated the code:

- The leftmost target component is processed first and generates the **ExpReport.xml** file.
- The component to the right of it is processed next and generates the **SecondXML.xml** file.

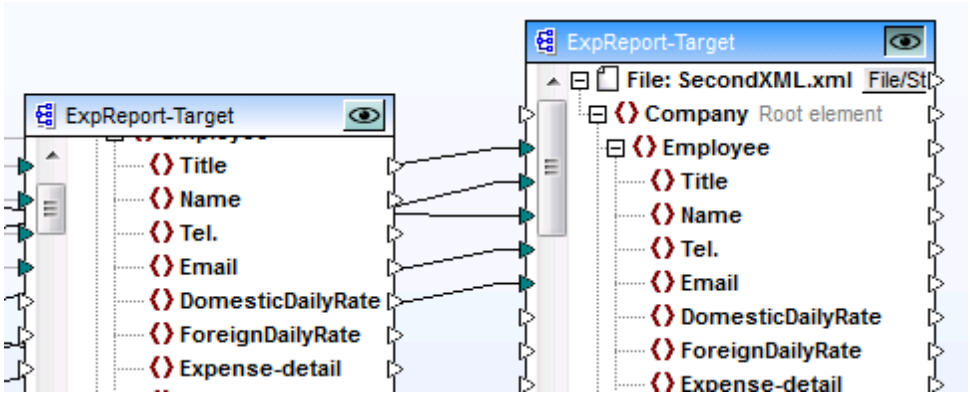
You can check that this is the case by opening the **DoTransform.bat** file (in the output folder you specified) and see the sequence the output files are generated. **ExpReport-Target.xml** is the first output to be generated by the batch file, and **SecondXML.xml** the second.

```
@echo off

RaptorXML xslt --xslt-version=2 --
input="C:\Users\me\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\m
f-ExpReport.xml" --
output="C:\Users\me\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\
ExpReport-Target.xml" %* "MappingMapToExpReport-Target.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --
input="C:\Users\me\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\m
f-ExpReport.xml" --
output="C:\Users\me\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\
SecondXML.xml" %* "MappingMapToExpReport-Target2.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

Changing the mapping processing sequence:

1. Click the left target component and move it below the one at right.



2. Regenerate your code and take a look at the **DoTransform.bat** file.

```
@echo off

RaptorXML xslt --xslt-version=2 --
input="C:\Users\alp\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\
mf-ExpReport.xml" --
output="C:\Users\alp\Documents\Altova\MapForce2013\MapForceExamples\Tutorial
\SecondXML.xml" %* "MappingMapToExpReport-Target.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --
input="C:\Users\alp\Documents\Altova\MapForce2013\MapForceExamples\Tutorial\
mf-ExpReport.xml" --
output="C:\Users\alp\Documents\Altova\MapForce2013\MapForceExamples\Tutorial
\ExpReport-Target.xml" %* "MappingMapToExpReport-Target2.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

SecondXML.xml is now the first output to be generated by the batch file, and **ExpReport-Target.xml** the second.

Chained mappings

The same processing sequence as described above is followed for chained mappings. The chained mapping group is taken as one unit however. Repositioning the intermediate or final target component of a single chained mapping has no effect on the processing sequence.

Only if multiple "chains" or multiple target components exist in a mapping does the position of the **final** target components of each group determine which is processed first.

- If two final target components have the same vertical position, then the leftmost takes precedence.
- If two final target component have the same horizontal position, then the highest takes precedence.
- In the unlikely event that components have the exact same position, then an unique internal component ID is automatically used, which guarantees a well-defined order but which cannot be changed.

 © 2015 Altova GmbH

Chapter 6

Data Sources and Targets

6 Data Sources and Targets

This section describes the various source and target component types that MapForce can map from/to.

- XML and XML schema
- HL7 v3.x to/from XML schema

© 2015 Altova GmbH

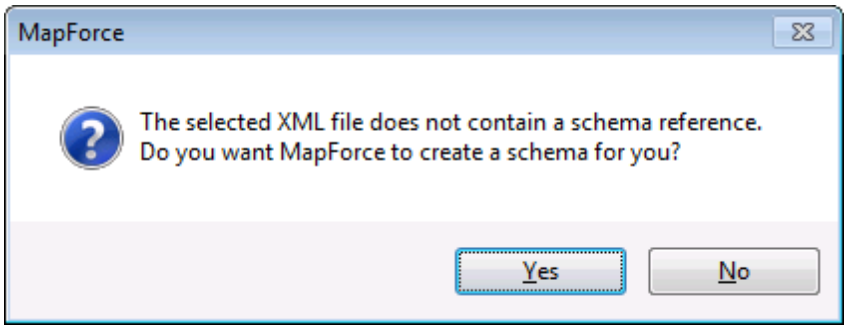
6.1 XML and XML schema

In the introductory part of this documentation, you have seen examples of simple mappings that use XML and XML schema files as source or target components. This section provides further information about using XML components in your mappings. It includes the following topics:

- XML Component Settings
- Using DTDs as "schema" components
- Derived XML Schema types - mapping to
- QName support
- Nil Values / Nillable
- Comments and Processing Instructions
- CData sections
- Wildcards - xs:any

6.1.1 Generating an XML Schema

MapForce can automatically generate an XML schema based on an existing XML file if the XML Schema is not available. When you add to the mapping area an XML file without a schema, the following dialog box appears.

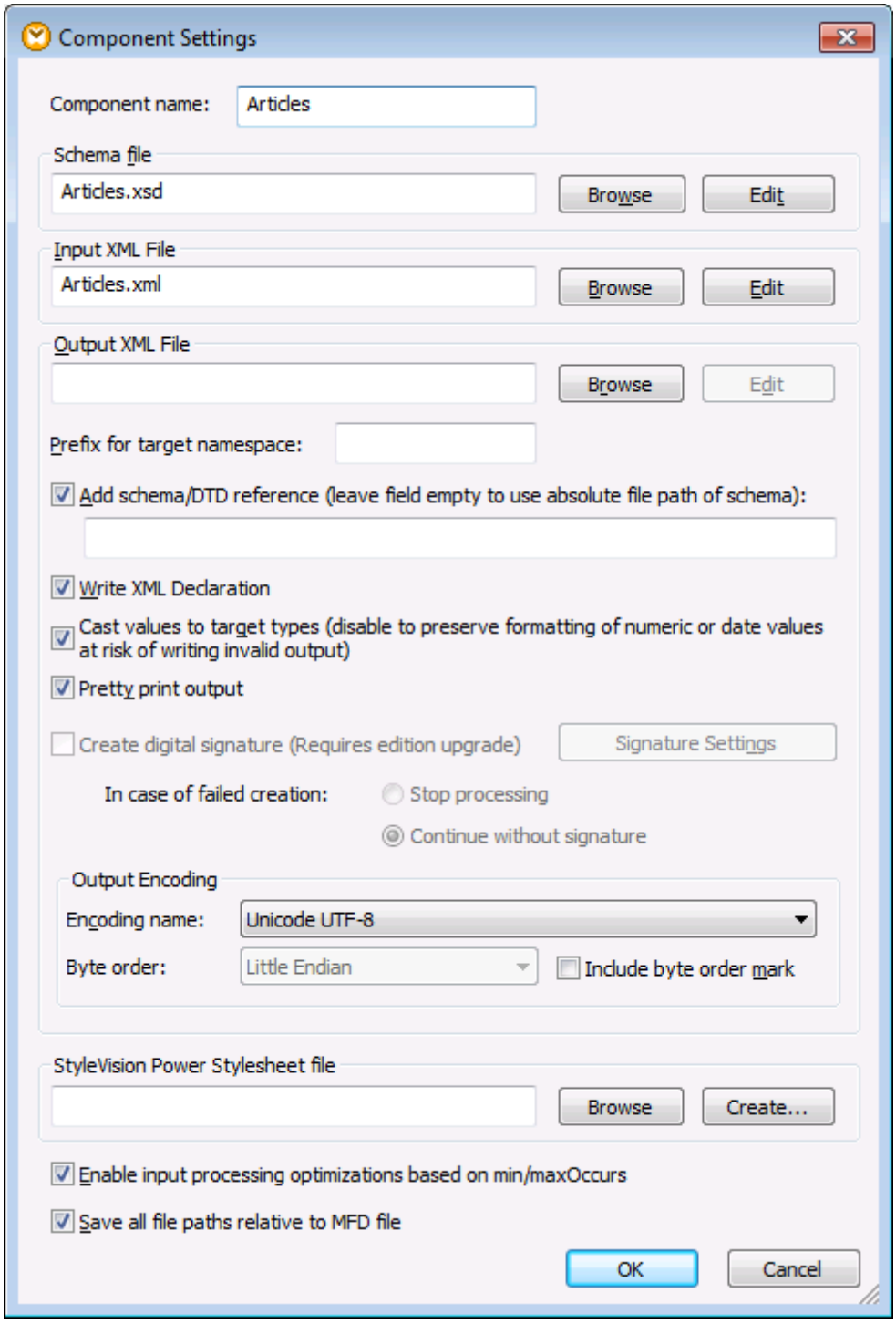


When MapForce generates a schema from an XML file, data types for elements/attributes must be inferred from the XML instance document and may not be exactly what you expect. It is recommended that you check whether the generated schema is an accurate representation of the instance data.

6.1.2 XML Component Settings

After you add an XML component to the mapping area, you can configure the settings applicable to it from the Component Settings dialog box. You can open the Component settings dialog box in one of the following ways:

- Select the component on the mapping, and, on the **Component** menu, click **Properties**.
- Double-click the component header.
- Right-click the component header, and then click **Properties**.



XML Component Settings dialog box

The available settings are as follows.

Component name	The component name is automatically generated when you

 © 2015 Altova GmbH

	create the component. You can however change the name at any time. If the component name was automatically generated and you select an instance file after that, MapForce will prompt you to optionally update the component name as well. The component name can contain spaces (for example, "Source XML File") and full stop characters (for example, "Orders.EDI"). The component name may not contain slashes, backslashes, colons, double quotes, leading or trailing spaces. In general, be aware of the following implications when changing the name of the component: • If you intend to deploy the mapping to FlowForce Server, the component name must be unique. • It is recommended to use only characters that can be entered at the command line. National characters may have different encodings in Windows and at the command line.
Schema file	Specifies the name or path of the XML schema file used by MapForce to validate and map data. To change the schema file, click **Browse** and select the new file. To edit the file in XMLSpy, click **Edit**.
Input XML file	Specifies the XML instance file from which MapForce will read data. This field is meaningful for a source component and is filled when you first create the component and assign to it an XML instance file. In a source component, the instance file name is also used to detect the XML root element and the referenced schema, and to validate against the selected schema. To change the location of the file, click **Browse** and select the new file. To edit the file in XMLSpy, click **Edit**.
Output XML file	Specifies the XML instance file to which MapForce will write data. This field is meaningful for a target component. To change the location of the file, click **Browse** and select the new file. To edit the file in XMLSpy, click **Edit**.
Prefix for target namespace	Allows you to enter a prefix for the target namespace. Ensure that the target namespace is defined in the target schema, before assigning the prefix.
Add schema/DTD reference	Adds the path of the referenced XML Schema file to the root element of the XML output. The path of the schema entered in this field is written into the generated target instance files in the `xsi:schemaLocation` attribute, or into the `DOCTYPE`

<table>
<tr><td></td><td>declaration if a DTD is used.

Entering a path in this field allows you to define where the schema file referenced by the XML instance file is to be located. This ensures that the output instance can be validated at the mapping destination when the mapping is executed. You can enter an http:// address as well as an absolute or relative path in this field.

Deactivating this option allows you to decouple the XML instance from the referenced XML Schema or DTD (for example, if you want to send the resulting XML output to someone who does not have access to the underlying XML Schema).</td></tr>
<tr><td>Write XML declaration</td><td>This option enables you to suppress the XML declaration from the generated output. By default, the option is enabled, meaning that the XML declaration is written to the output.

This feature is supported as follows in MapForce target languages and execution engines.

<table>
<tr><th>Target language / Execution engine</th><th>When output is a file</th><th>When output is a string</th></tr>
<tr><td>XSLT, XQuery</td><td>Yes</td><td>No</td></tr>
</table>
</td></tr>
<tr><td>Cast values to target types</td><td>Allows you to define if the target XML schema types should be used when mapping, or if all data mapped to the target component should be treated as string values. By default, this setting is enabled.

Deactivating this option allows you to retain the precise formatting of values. For example, this is useful to satisfy a pattern facet in a schema that requires a specific number of decimal digits in a numeric value.

You can use mapping functions to format the number as a string in the required format, and then map this string to the target.

Note that disabling this option will also disable the detection of invalid values, e.g. writing letters into numeric fields.</td></tr>
<tr><td>Pretty print output</td><td>Reformats the output XML document to give it a structured look. Each child node is offset from its parent by a single tab character.</td></tr>
<tr><td>Output Encoding</td><td>Allows you specify the following settings of the output instance file:</td></tr>
</table>

 © 2015 Altova GmbH

	• Encoding name • Byte order • Whether the byte order mark (BOM) character should be included. By default, any new components have the encoding defined in the **Default encoding for new components** option. You can access this option from **Tools | Options**, General tab. If the mapping generates XSLT 1.0/2.0, activating the **Byte Order Mark** check box does not have any effect**,** as these languages do not support Byte Order Marks.
StyleVision Power Stylesheet file	This option allows you to select or create an Altova StyleVision stylesheet file. Such files enable you to output data from the XML instance file to a variety of formats suitable for reporting, such as HTML, RTF, and others. See also Using Relative Paths on a Component.
Enable input processing optimizations based on min/ maxOccurs	This option allows special handling for sequences that are known to contain exactly one item, such as required attributes or child elements with `minOccurs` and `maxOccurs="1"`. In this case, the first item of the sequence is extracted, then the item is directly processed as an atomic value (and not as a sequence). If the input data is **not valid** against the schema, an empty sequence might be encountered in a mapping, which stops the mapping with an error message. To allow the processing of such **invalid input**, disable this check box.
Save all file paths relative to MFD file	When this option is enabled, MapForce saves the file paths displayed on the Component Settings dialog box relative to the location of the MapForce Design (.mfd) file. See also Using Relative Paths on a Component.

6.1.3 Using DTDs as "Schema" Components

Starting with MapForce 2006 SP2, namespace-aware DTDs are supported for source and target components. The namespace-URIs are extracted from the DTD "xmlns"-attribute declarations, to make mappings possible.

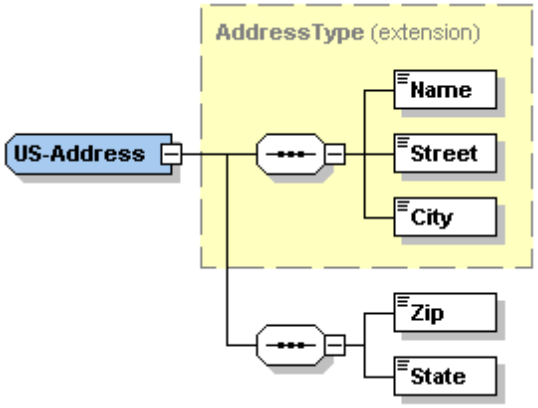
However, some DTDs contain xmlns*-attribute declarations without namespace-URIs (for example, DTDs used by StyleVision). Such DTDs have to be extended to make them useable in MapForce. Specifically, you can make such DTDs useable by defining the xmlns-attribute with the namespace-URI, as shown below:

```
<!ATTLIST fo:root
   xmlns:fo CDATA #FIXED 'http://www.w3.org/1999/XSL/Format'
   ...
>
```

6.1.4 Derived XML Schema Types

MapForce supports the mapping to/from derived types of a complex type. Derived types are complex types of an XML Schema that use the **xsi:type** attribute to identify the specific derived types.

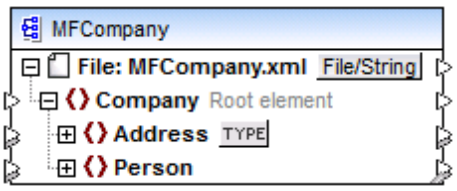
The screenshot below shows the definition of a derived type called `US-Address`, in XMLSpy. The base type (or originating complex type) is `AddressType`. Two extra elements were added to create the derived type `US-Address`: `Zip` and `State`.



Sample derived type (XMLSpy schema view)

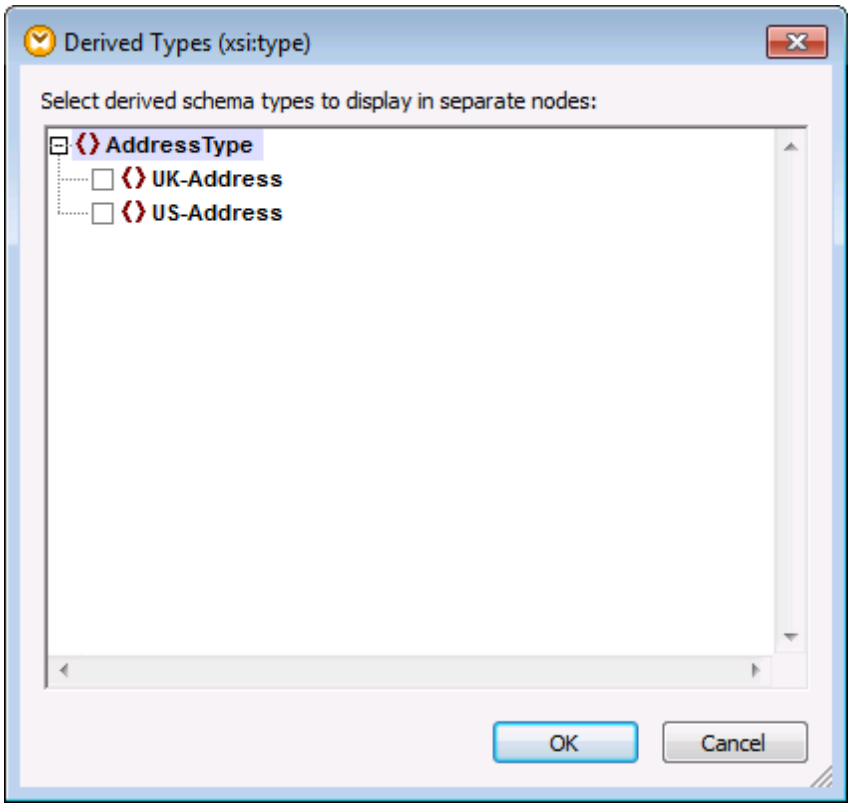
The following example shows you how to map data to or from derived XML schema types.

1. On the **Insert** menu, click **XML Schema/File**, and open the following XML Schema: **<Documents>\Altova\MapForce2016\MapForceExamples\Tutorial\MFCompany.xsd**.
2. When prompted to supply an instance file, click **Skip**, and then select `Company` as the root element.

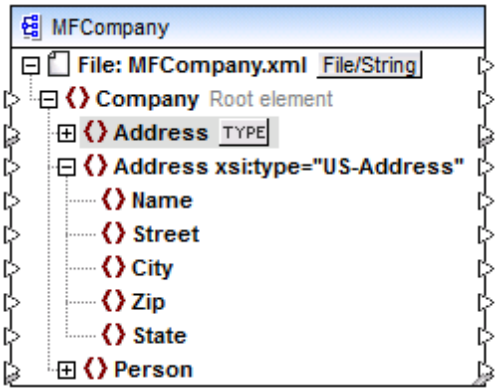


3. Click the TYPE button next to the `Address` element. This button indicates that derived types exist for this element in the schema.

 © 2015 Altova GmbH



4. Select the check box next to the derived type you want to use (`US-Address`, in this case), and confirm with OK. A new element `Address xsi:type="US-Address"` has been added to the component.



You can now map data to or from the `US-Address` derived type.

Note that you can also include multiple derived types by selecting them in the Derived Types dialog box. In this case, each would have its own `xsi:type` element in the component.

6.1.5 QNames

MapForce resolves QName (qualified name) prefixes (http://www.w3.org/TR/xml-names/#ns-qualnames) when reading data from XML files at mapping execution run-time.

QNames are used to reference and abbreviate namespace URIs in XML instance documents. There are two types of QNames: Prefixed and Unprefixed QNames.

PrefixedName	Prefix ':'	LocalPart
UnPrefixedName		LocalPart where LocalPart is an Element or Attribute name.

For example, in the listing below, `<x:p/>` is a QName, where:

- the prefix "x" is an abbreviation of the namespace "**http://myCompany.com**".
- `p` is the local part.

```
<?xml version='1.0'?>
<doc xmlns:x="http://myCompany.com">
   <x:p/>
</doc>
```

MapForce also includes several QName-related functions in the **xpath2 | qname-related functions** library. The **lang | QName** library (available in the MapForce Professional and Enterprise Editions) additionally includes QName-related functions that can be used in code generation languages and the BUILT-IN transformation language.

6.1.6 Nil Values / Nillable

The XML Schema specification allows for an element to be valid without content if the `nillable="true"` attribute has been defined for that specific element in the schema. In the instance XML document, you can then indicate that the value of an element is nil by adding the `xsi:nil="true"` attribute to it. This section describes how MapForce handles nil elements in source and target components.

'xsi:nil' versus 'nillable'

The `xsi:nil="true"` attribute is defined in the XML **instance** document.

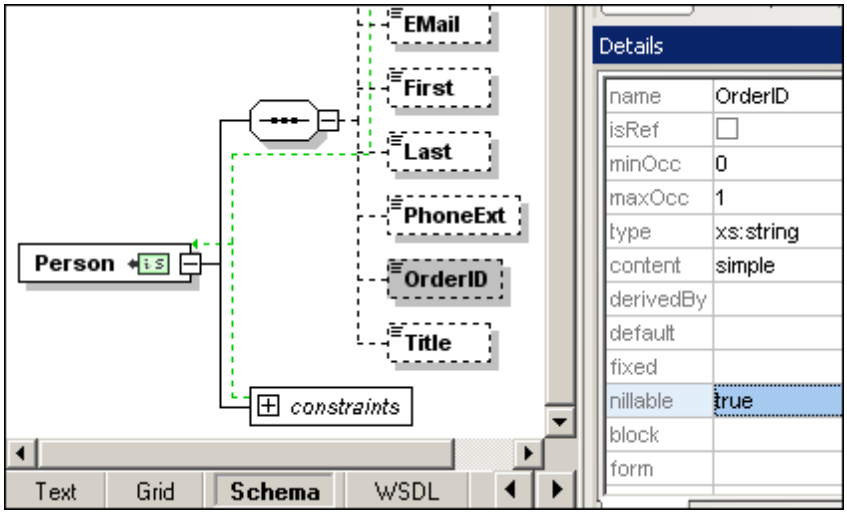
© 2015 Altova GmbH

```
14   <Person>
15     <PrimaryKey>2</PrimaryKey>
16     <ForeignKey>1</ForeignKey>
17     <EMail>biff@amail.com</EMail>
18     <First>biff</First>
19     <Last>bander</Last>
20     <PhoneExt>22</PhoneExt>
21     <OrderID xsi:nil="true"/>
22     <Title>IT services</Title>
23   </Person>
```

The `xsi:nil="true"` attribute indicates that, although the element exists, it has no content. Note that the `xsi:nil="true"` attribute applies to element values, and not to attribute values. An element with `xsi:nil="true"` may still have other attributes, even if it does not have content.

The `xsi:nil` attribute is not displayed explicitly in the MapForce graphical mapping, because it is handled automatically in most cases. Specifically, a "nilled" node (one that has the `xsi:nil="true"` attribute) exists, but its content does not exist.

The `nillable="true"` attribute is defined in the XML **schema**. In MapForce, it can be present in both the source and target components.



Nillable elements as mapping source

MapForce checks the `xsi:nil` attribute automatically, whenever a mapping reads data from nilled XML elements. If the value of `xsi:nil` is `true`, the content will be treated as non-existent.

When you create a **Target-driven** mapping from a nillable source element to a nillable target element with **simple content** (a single value with optional attributes, but without child elements), where `xsi:nil` is set on a source element, MapForce adds the `xsi:nil` attribute to the target element (for example, `<OrderID xsi:nil="true"/>`).

When you create a **Copy-All** mapping from a nillable source element to a nillable target element, where `xsi:nil` is set on a source element, MapForce adds the `xsi:nil` attribute to the target element (for example, `<OrderID xsi:nil="true"/>`).

To check explicitly whether a source element has the `xsi:nil` attribute set to `true`, use the **is-xsi-nil** function. It returns TRUE for nilled elements and FALSE for other nodes.

To substitute a nilled (non-existing) source element value with something specific, use the **substitute-missing** function.

Notes:

- Connecting the **exists** function to a nilled source element returns TRUE, since the element node actually exists, even if it has no content.
- Using functions that expect simple values (such as **multiply** and **concat**) on elements where `xsi:nil` has been set does not yield a result, as no element content is present and no value can be extracted. These functions behave as if the source node did not exist.

Nillable elements as mapping target

When you create a **Target-driven** mapping from a nillable source element to a nillable target element with **simple content** (a single value with optional additional attributes, but without child elements), where `xsi:nil` is set on a source element, MapForce inserts the `xsi:nil` attribute into the target element (for example, `<OrderID xsi:nil="true"/>`). If the `xsi:nil="true"` attribute has not been set in the XML source element, then the element content is mapped to the target element in the usual fashion.

When mapping to a nillable target element with **complex type** (with child elements), the `xsi:nil` attribute will **not** be written automatically, because MapForce cannot know at the time of writing the element's attributes if any child elements will follow. For such cases, define a **Copy-All** connection to copy the `xsi:nil` attribute from the source element.

When mapping an **empty sequence** to a target element, the element will not be created at all, independent of its nillable designation.

To force the creation of an empty target element with `xsi:nil` set to `true`, connect the **set-xsi-nil** function directly to the target element. This works for target elements with simple and complex types.

If the node has simple type, use the **substitute-missing-with-xsi-nil** function to insert `xsi:nil` in the target if no value from your mapping source is available. This can happen if the source node does not exist at all, or if a calculation (for example, multiply) involved a nilled source node and therefore yielded no result.

Note:

- Functions which generate `xsi:nil` cannot be passed through functions or components which only operate on values (such as the **if-else** function).

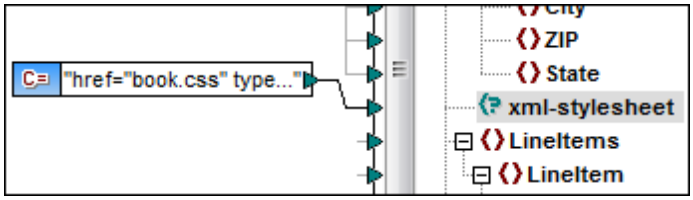
© 2015 Altova GmbH

6.1.7 Comments and Processing Instructions

Comments and Processing Instructions can be inserted into target XML components. Processing instructions are used to pass information to applications that further process XML documents. Note that Comments and Processing instructions cannot be defined for nodes that are part of a copy-all mapped group.

To insert a Processing Instruction:

1. Right-click an element in the target component and select Comment/Processing Instruction, then one of the Processing Instruction options from the menu (Before, After)
2. Enter the Processing Instruction (target) name in the dialog and press OK to confirm, e.g. xml-stylesheet.
   This adds a node of this name to the component tree.

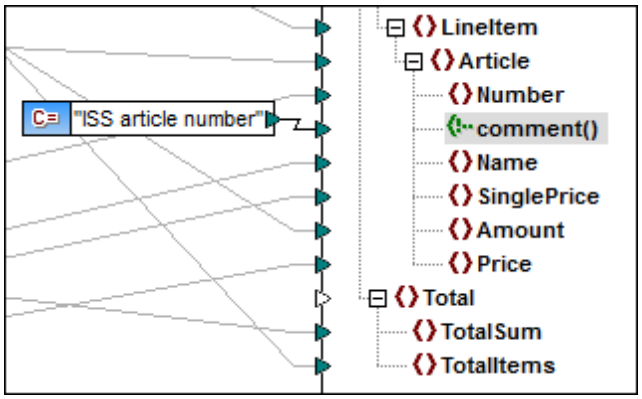


3. You can use, for example, a constant component to supply the value of the Processing Instruction attribute, e.g. `href="book.css" type="text/css"`.

Note:
Multiple Processing Instructions can be added before or after any element in the target component.

To insert a comment:

1. Right-click an element in the target component and select Comment/Processing Instruction, then one of the Comment options from the menu (Before, After).



   This adds the comment node (`<!--comment()`) to the component tree.
2. Use a constant component to supply the comment text, or connect a source node to the comment node.

Note:
Only one comment can be added before and after a single target node. To create multiple comments, use the duplicate input function.

To delete a Comment/Processing Instruction:

- Right-click the respective node, select Comment/Processing Instruction, then select Delete Comment/Processing Instruction from the flyout menu.

6.1.8 CDATA Sections

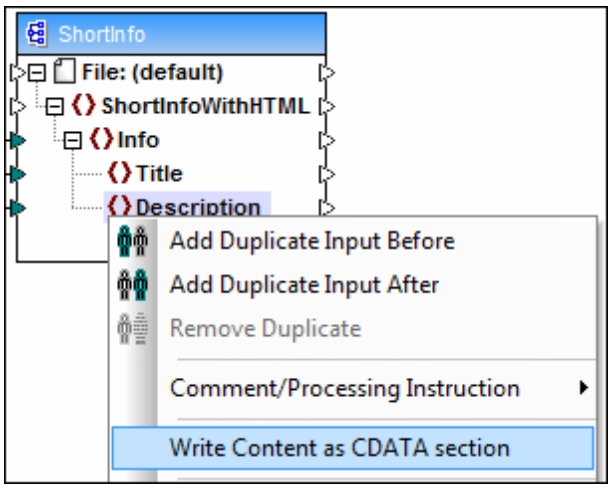
CDATA sections are used to escape blocks of text containing characters which would normally be interpreted as markup. CDATA sections start with "<![CDATA[" and end with the "]]>".

Target nodes can now write the input data that they receive as CDATA sections. The target node components can be:

- XML data
- XML data embedded in database fields
- XML child elements of typed dimensions in an XBRL target

To create a CDATA section:

1. Right-click the target node that you want to define as the CDATA section and select "Write Content as CDATA section".



A prompt appears warning you that the input data should not contain the CDATA section close delimiter ']]>', click OK to close the prompt.
The [C.. icon shown below the element tag shows that this node is now defined as a CDATA section.

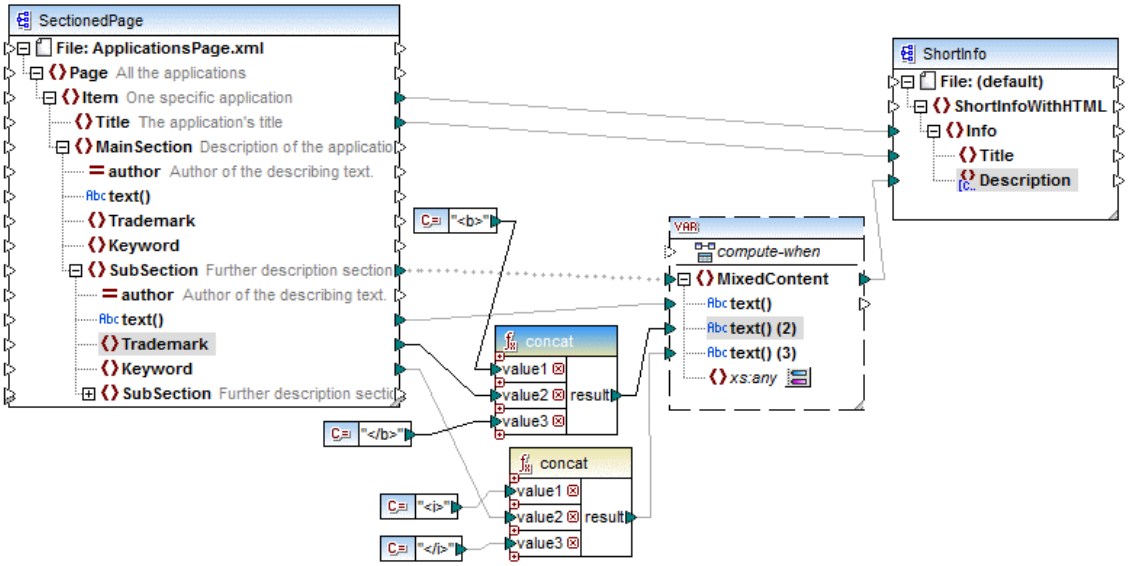
Note:
CDATA sections can also be defined on duplicate nodes, and xsi:type nodes.

 © 2015 Altova GmbH

Example:
The **HTMLinCDATA.mfd** mapping file available in the ...\MapForceExamples folder shows an example of where CDATA sections can be very useful.

In this example:

- Bold start () and end () tags are added to the content of the **Trademark** source element.
- Italic start () and end () tags are added to the content of the **Keyword** source element.
- The resulting data is passed on to duplicate **text()** nodes in the order that they appear in the source document, due to the fact the Subsection element connector, has been defined as a Source Driven (Mixed content) node.
- The output of the MixedContent node is then passed on to the **Description** node in the ShortInfo target component, which has been defined as a CDATA section.

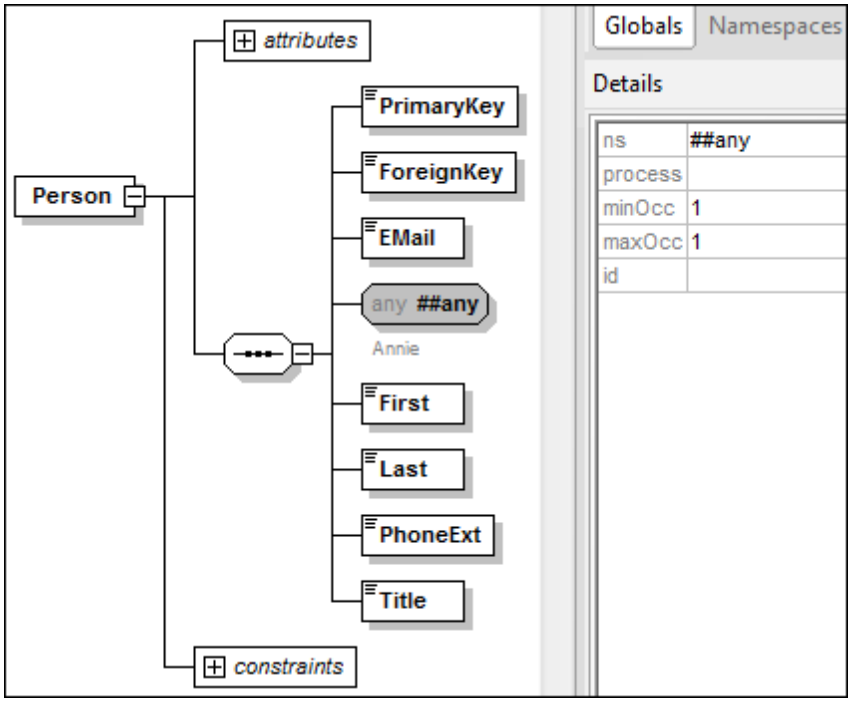


Clicking the Output button shows the CDATA section containing the marked-up text.

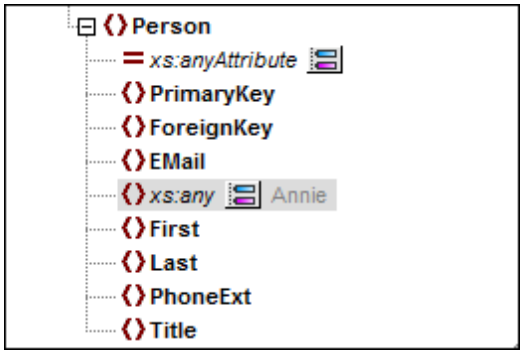
```
7    <Info>
8      <Title>MapForce</Title>
9      <Description><![CDATA[Altova <b>MapForce</b> 2014 Enterprise Edition is the premier <i>XML</i>
/ <i>database</i> / <i>flat file</i> / <i>EDI</i> data mapping tool that auto-generates mapping code in
<i>XSLT</i> 1.0/2.0, <i>XQuery</i>, <i>Java</i>, <i>C++</i> and <i>C#</i>. It is the definitive tool for
data integration and information leverage.]]></Description>
10   </Info>
```

6.1.9 Wildcards - xs:any / xs:anyAttribute

The wildcards xs:any (and xs:anyAttribute) allow you to use any elements/attributes from schemas . The screenshot shows the "any" element in the Schema view of XMLSpy.

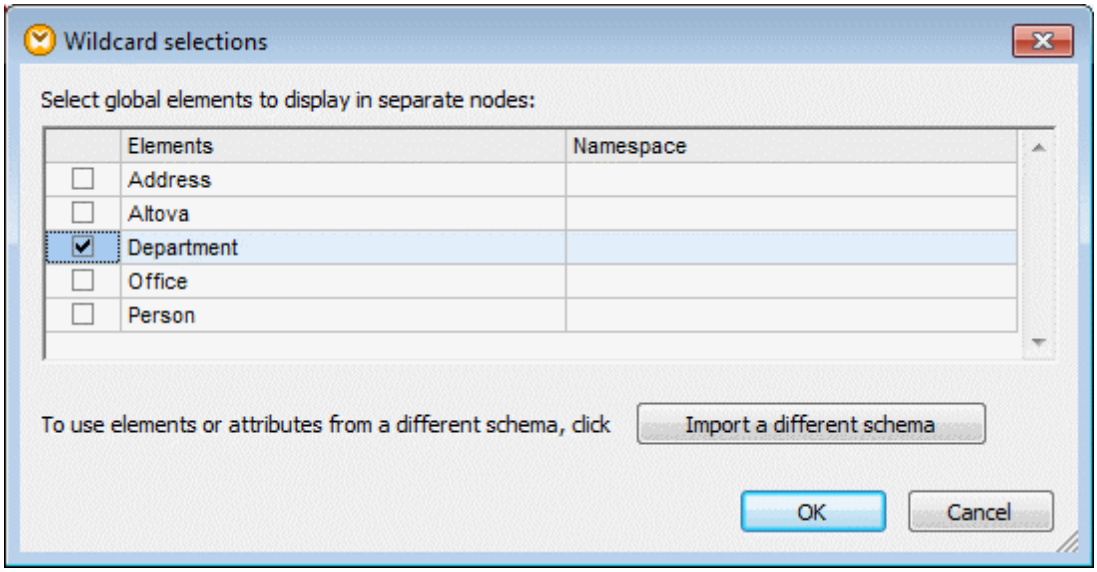


In MapForce the schema structure is shown as below with a "selection" button to the right of the xs:any element (and xs:anyAttribute).

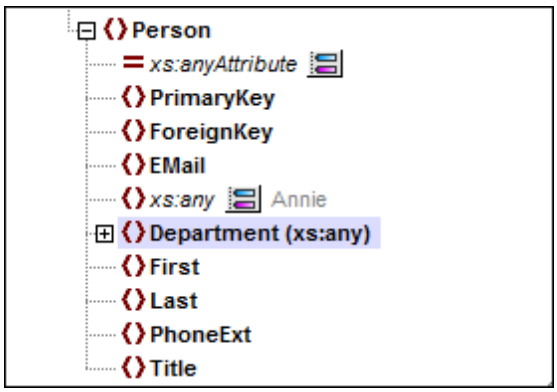


Clicking the xs:any selection button opens the "Wildcard selections" dialog box. The entries in this listbox show the global elements/attributes declared in the current schema.

 © 2015 Altova GmbH



Clicking one, or more of the check boxes and confirming with OK, inserts that element/attribute (and any other child nodes) into the component at that position.



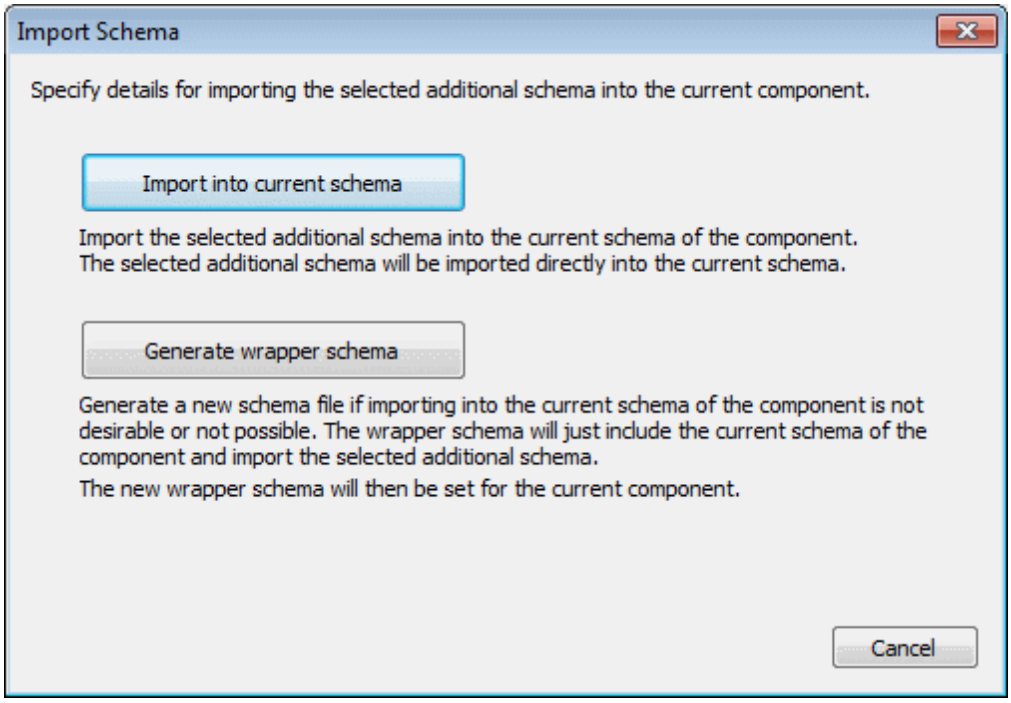
You can now map to/from these nodes as with any other element.

To remove a wildcard element:

- Click the selection button, then deselect the global elements.

To use elements from a different schema:

1. Click the "Import a different schema" button in the "Wildcard selections" dialog box.
2. Select the schema you want to import the elements from.
   You can now define if you want to import the other schema into the currently open one, or generate a new "wrapper" schema which contains references to both schemas.

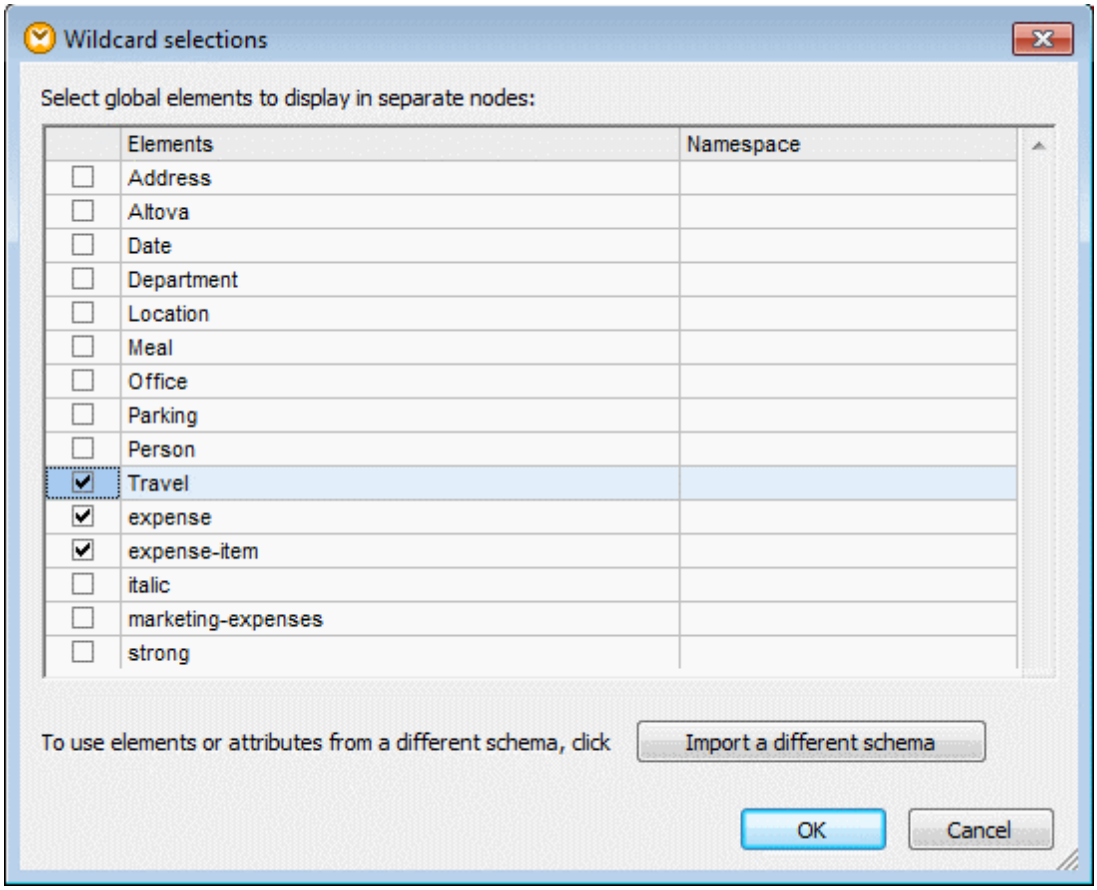

Import Schema
Specify details for importing the selected additional schema into the current component.
Import into current schema
Import the selected additional schema into the current schema of the component.
The selected additional schema will be imported directly into the current schema.
Generate wrapper schema
Generate a new schema file if importing into the current schema of the component is not desirable or not possible. The wrapper schema will just include the current schema of the component and import the selected additional schema.
The new wrapper schema will then be set for the current component.
Cancel

3. Click the button to select which of the two options you want.

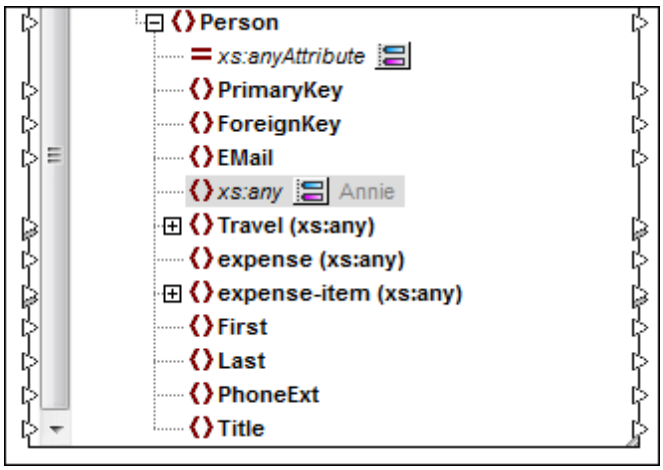
Import into current schema

The screenshot shows the global elements available if the imported schema is the **HasExpenses.xsd** file available in the ...\MapForceExamples folder. Three expense related elements have been selected.

 © 2015 Altova GmbH



Having clicked OK, each of the elements (and any child elements they may have) are added to the component below the xs:any wildcard node.



Each of the imported nodes has a (xs:any) annotation to show that it has been imported.

Note:

Importing into the current schema, **overwrites** the schema at that location. You must therefore have the user rights to do so. A remote schema that you might have opened using the "Switch to URL" button cannot be overwritten, and cannot be imported in this way.

Generate wrapper schema

1. Click the "Generate wrapper schema" button in the "Wildcard selections" dialog box, enter the name of the new wrapper schema and click Save.
   A default name is supplied in the form XXXX-wrapper.xsd. This new wrapper schema will include the current schema and import the other one.

   A prompt appears asking if you want the Component Setting schema location to reference the wrapper schema, or if you want to change it to reference the previous main schema.
2. Click No to keep the reference to the wrapper schema, or click Yes to change the schema location to the previous main schema.
   The imported nodes are shown as before in the component.

Note:
Using the generate wrapper option allows you to use remote schemas, with a URL, and add elements from another schema to the current component.

6.1.10 Mapping to the Root Element

An XML document may have only one root element. Therefore, when drawing a connection to the root element of the target Schema/XML file, make sure that the connection passes only one item to the target XML component. Otherwise, you will create well-formed, but not valid, XML files.

Use a filter component (see Filtering) to limit the mapped data to a single element or record.

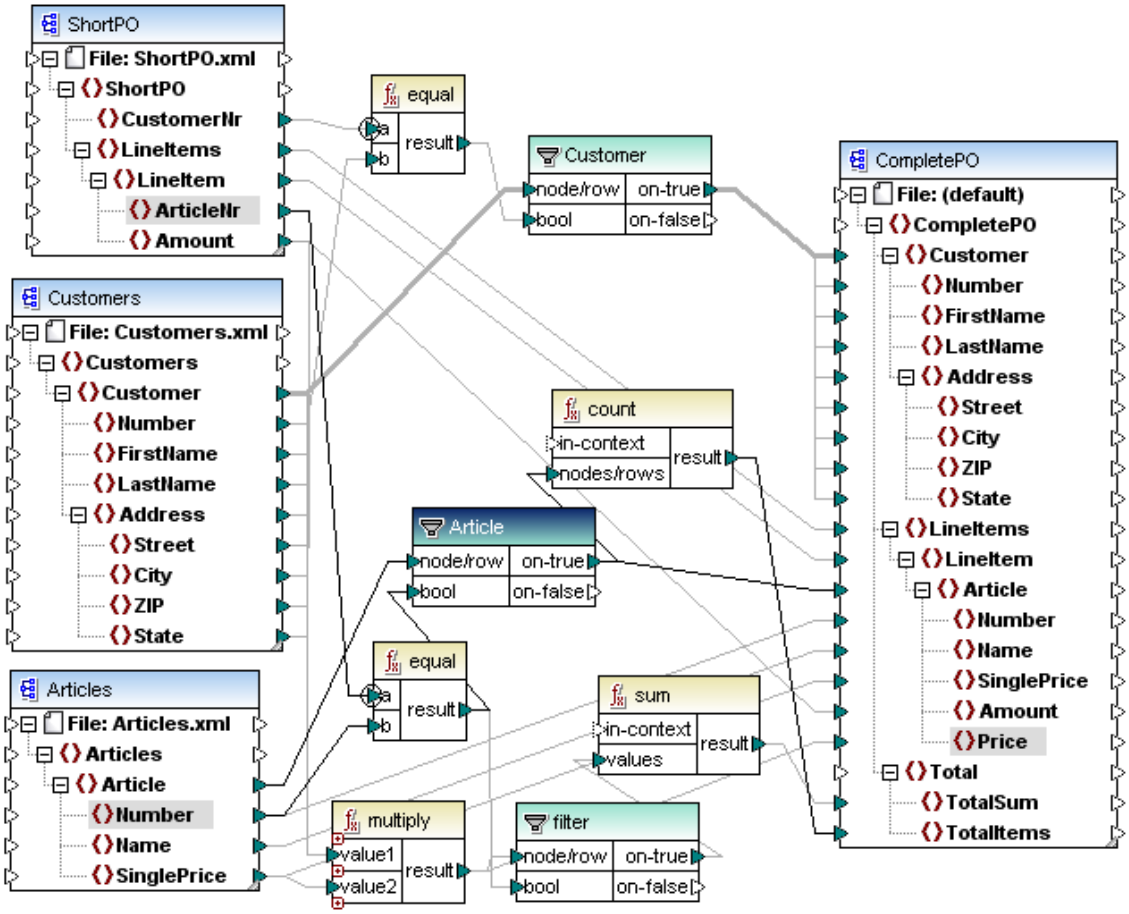
6.1.11 Merging Data from Multiple Schemas

MapForce allows you to merge multiple files into a single target file.

This example merges multiple source components with different schemas to a target schema. To merge an arbitrary number of files using the same schema, see Processing Multiple Input or Output Files Dynamically.

The **CompletePO.mfd** file available in the **...\MapForceExamples** folder shows how three XML files are merged into one purchasing order XML file.

© 2015 Altova GmbH



Note that multiple source component data are combined into one target XML file - CompletePO

- **ShortPO** is a schema with an associated XML instance file and contains only customer number and article data, i.e. Line item, number and amount.
- **Customers** is a schema with an associated XML instance file and contains customer number and customer information details, i.e. Name and Address info. (There is only one customer in this file with the Customer number of 3)
- **Articles** is a schema with an associated XML instance and contains article data, i.e. article name number and price.
- **CompletePO** is a schema file without an instance file as all the data is supplied by the three XML instance files. The hierarchical structure of this file makes it possible to merge and output all XML data.

This schema file has to be created in an XML editor such as XMLSpy, it is not generated by MapForce (although it would be possible to create if you had an CompletePO.xml instance file).

The stucture of CompletePO is a combination of the source XML file structures.

The **filter** component (Customer) is used to find/filter the data where the customer numbers are identical in both the ShortPO and Customers XML files, and pass on the associated data to the target CompletePO component.

- The **CustomerNr** in ShortPO is compared with the **Number** in Customers using the "equal" function.
- As ShortPO only contains one customer (number 3), only customer and article data for

customer number 3, can be passed on to the filter component.

- The **node/row** parameter, of the filter component, passes on the **Customer** data to "on-true" when the bool parameter is true, i.e. when the same number has been found, in this case customer number 3.
- The rest of the customer and article data are passed on to the target schema through the two other filter components.

© 2015 Altova GmbH

6.2 HL7 v3.x to/from XML schema mapping

Support for HL7 version **3.x** is automatically included in MapForce 2016 as it is XML based.

A separate installer for the HL7 V2.2 - V2.5.1 XML Schemas and configuration files is available on the Libraries page of the Altova website (http://www.altova.com/components_mapforce.html) Select the Custom Setup in the installer, to only install the HL7 V3 components and XML Schemas.

Location of HL7 XML Schemas after installation:

32-bit MapForce on 32-bit operating system, or 64-bit MapForce on 64-bit operating system	C:\Program Files\Altova\Common2016\Schemas\hl7v3
32-bit MapForce on 64-bit operating system	C:\Program Files(x86)\Altova\Common2016\Schemas\hl7v3

HL7 documents can be used as source and target components in MapForce. This data can also be mapped to any number of XML schema components.

Chapter 7

Functions

7 Functions

Functions represent a powerful way to transform data according to your specific needs. This section provides instructions on working with functions (regardless if they are built-in to MapForce, defined by you, or reused from external sources). Use the following roadmap for quick access to specific tasks related to functions:

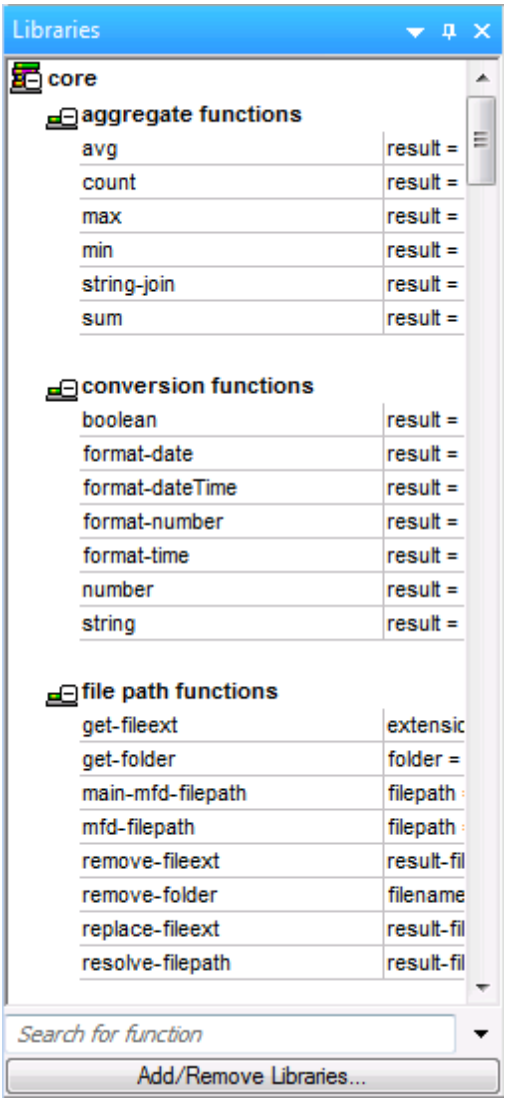
I want to...	Read this topic...
Learn how to work with functions in MapForce	Working with Functions
Learn how to change the priority of operations inside a mapping	Priority Context node/item
Create my own functions in MapForce	User-defined functions
Use external XSLT functions	Adding custom XSLT functions
View all built-in MapForce functions, or look up the description of a specific function.	Function Library Reference

© 2015 Altova GmbH

7.1 Working with Functions

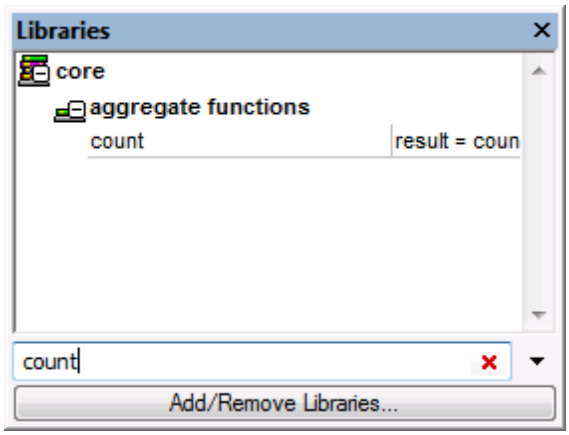
To use a function in a mapping:

1. Select the transformation language (see Selecting a transformation language).
2. Click the required function in the Libraries window and drag it to the mapping area.

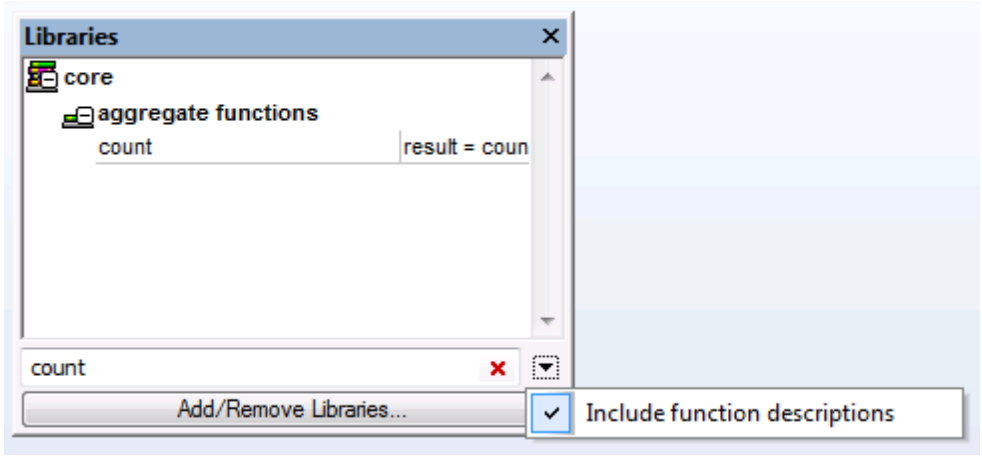


To search for a function in the Libraries window:

1. Start typing the function name in the text box located in the lower part of the Libraries window.



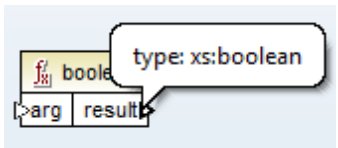
By default, MapForce searches by function name and description text. If you want to exclude the function description from the search, click the down-arrow and disable the **Include function descriptions** option.



To cancel the search, press the **Esc** key or click ×.

To view the data type of a function input or output argument:

1. Make sure that the **Show tips** toolbar button is enabled.
2. Move your mouse over the argument part of a function.

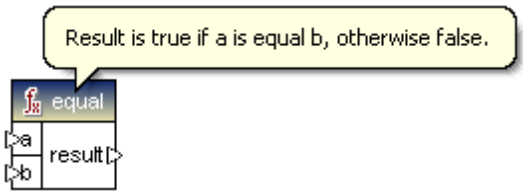


To view the description of a function:

1. Make sure that the **Show tips** toolbar button is enabled.

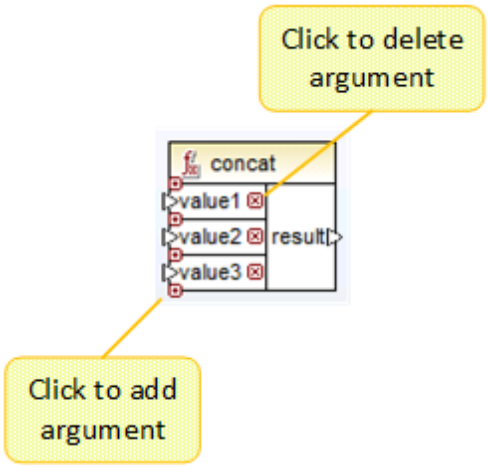
 © 2015 Altova GmbH

2. Move your mouse of the function (this works both in the Libraries pane and on the mapping area)



To add or delete function arguments (for functions where that is applicable):

- Click **Add parameter** (⊡) or **Delete parameter** (⊠) next to the parameter you want to add or delete, respectively.



Dropping a connection on the ⊡ symbol automatically adds the parameter and connects it.

7.1.1 Priority Context node/item

When applying a function to different items in a schema, MapForce needs to know what the context node will be. All other items are then processed relative to this one. This is achieved by designating the item (or node) as the priority context.

Priority-context is used to prioritize execution when mapping unrelated items.

Mappings are always executed top-down; if you loop/search through two tables then each loop is processed consecutively. When mapping unrelated elements, without setting the priority context, MapForce does not know which loop needs to be executed first, it therefore automatically selects the first table, or data source.

Solution:
Decide which source data is to be looped/searched first, and then set the priority context on the connector to that source data.

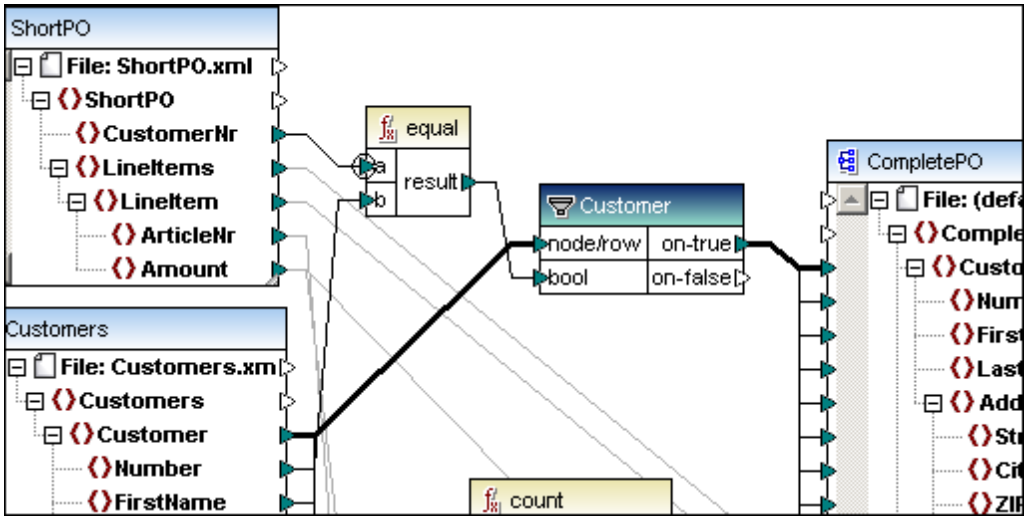
The **CompletePO.mfd** file available in the **...\MapForceExamples** folder, is shown below.

Please note that there are multiple source components in this example. **ShortPO, Customers, and Articles** are all schemas with associated XML instance files. The data from each, are then mapped to the CompletePO schema / XML file. The priority context icon, is enclosed in a circle as a visual indication.

- The **CustomerNr** in ShortPO is compared with the item **Number** in the Customers file.
- **CustomerNr** has been designated as the **priority context**, and is placed in the **a** parameter of the equal function.
- The **Customers** file is then searched (**once**) for the **same** number. The **b** parameter contains the Number item from the Customers file.
- If the number is found, then the result is passed to the **bool** parameter of the **filter** function.
- The **node/row** parameter passes on the **Customers** data to "on-true" when the bool parameter is true, i.e. when the same number has been found.
- The rest of the customer data is then passed on as: Number, FirstName, LastName items, are all connected to the corresponding items in the target schema.

Designating the **b** parameter of the equal function (i.e. item Number), as the **priority context** would cause:

- MapForce to load the first Number into the **b** parameter
- Check against the **CustomerNr** in **a,** if not equal,
- Load the next Number into **b**, check against a, and
- Iterate through every Number in the file while trying to find that number in ShortPO.

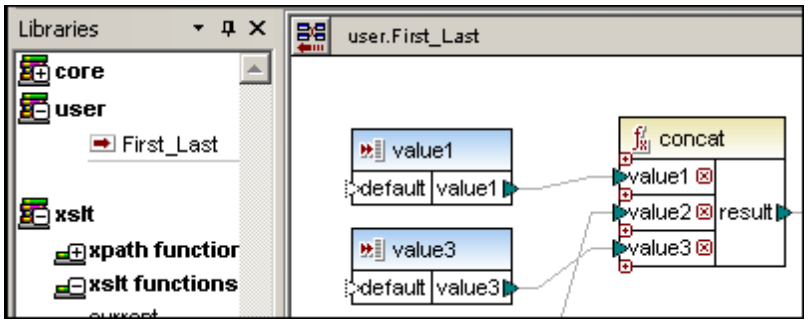


 © 2015 Altova GmbH

7.2 User-defined functions

MapForce allows you to create user-defined functions visually, in the same way as in the main mapping window.

These functions are then available as function entries in the Libraries window (e.g First_Last), and are used in the same way as the currently existing functions. This allows you to organize your mapping into separate building blocks, and reuse them in the same, or different mappings.



XSLT Selected

User-defined functions are stored in the ***.mfd** file, along with the main mapping.

A user-defined function uses **input** and **output components** to pass information from the main mapping (or another user-defined function) to the user-defined function and back.

User-defined functions can contain "local" source components (i.e that are within the user-defined function itself) such as XML schemas, which are useful when implementing lookup functions.

User-defined functions can contain any number of input and outputs where any of these can be in the form of: simple values, or XML nodes.

User-defined functions are useful when:

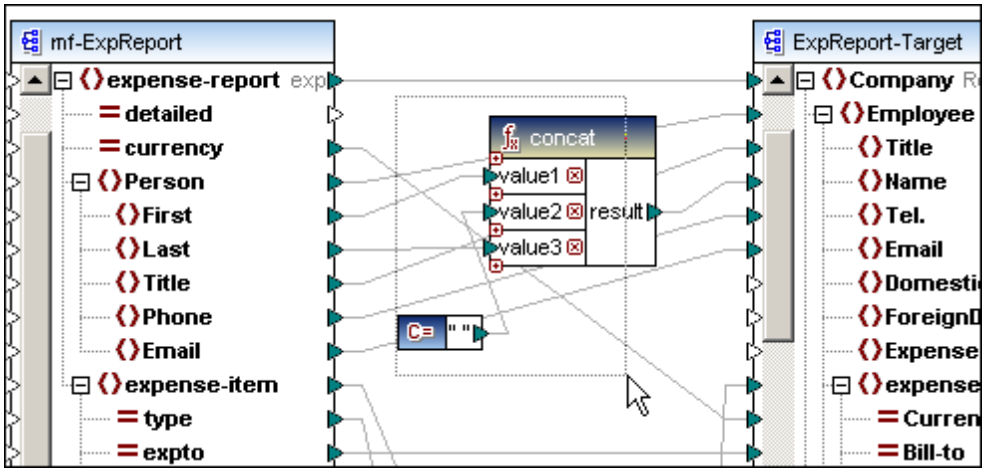
- combining multiple processing functions into a single component, e.g. for formatting a specific field or looking up a value
- reusing these components any number of times
- importing user-defined functions into other mappings (by loading the mapping file as a library)
- using inline functions to break down a complex mapping into smaller parts that can be edited individually
- mapping **recursive schemas** by creating recursive user-defined functions

User-defined functions can be either built from scratch, or from functions already available in the mapping tab.

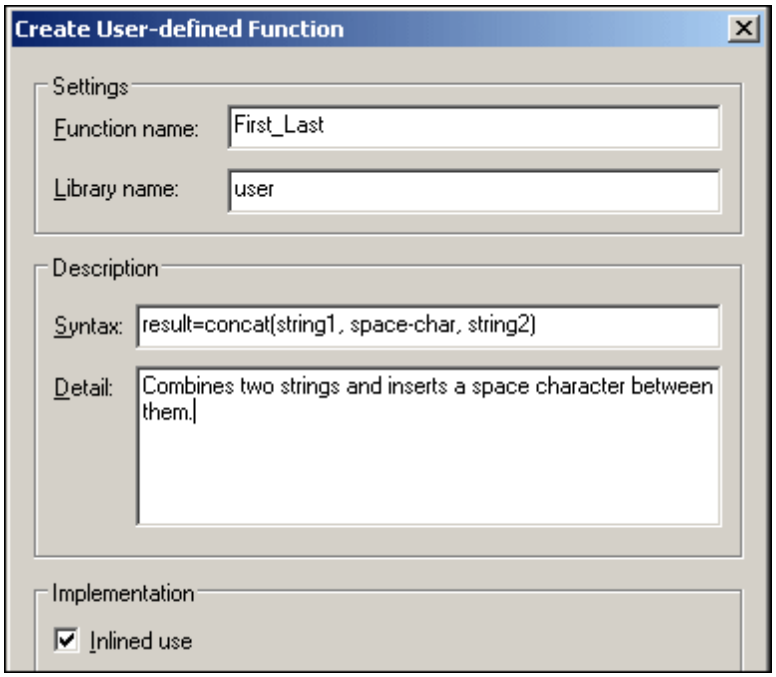
This example uses the **Tut-ExpReport.mfd** file available in the ...\MapForceExamples\Tutorial\ folder.

To create a user-defined function from existing components:

1. Drag to mark both the concat and the constant components (you can also hold down the CTRL key and click the functions individually).

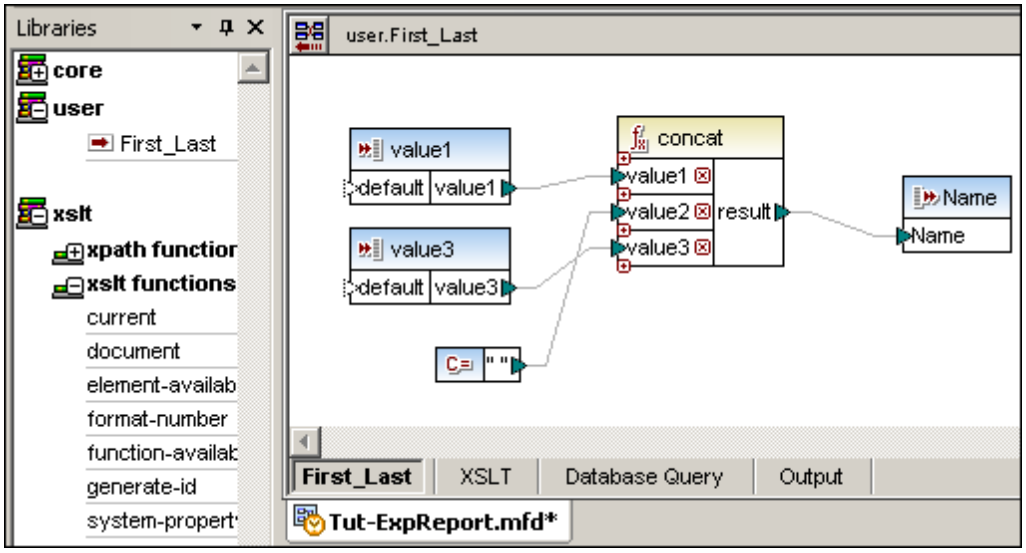


2. Select the menu option **Function | Create User-Defined Function from Selection**.
3. Enter the name of the new user-defined function (First_Last).
   Note: valid characters are: alphanumeric, a-z, A-Z, 0-9 as well as underscore "_", hyphen/dash "-" and colon ":".
4. Use the Syntax and Detail fields to add extra information on the new function, and click OK to confirm. The text you enter will appear as a tooltip when the cursor is placed over the function.
   The library name "user" is supplied as a default, you can of course define your own library name in this field.



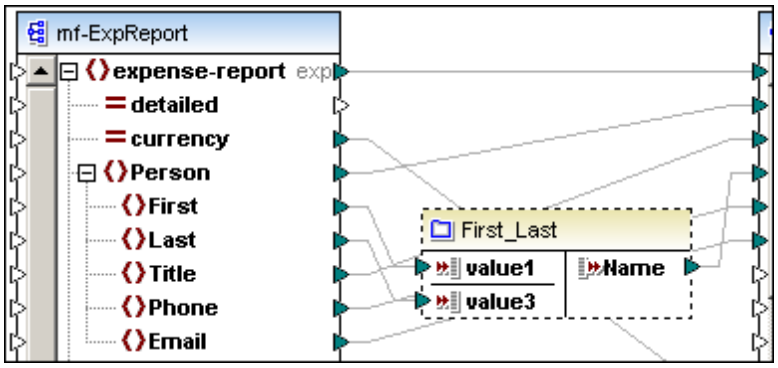
The individual elements that make up the function group appear in a tab with the function name. The new library "user" appears in the Libraries pane with the function name "First_Last" below it.

 © 2015 Altova GmbH



XSLT Selected

Click the Home button to return to the main mapping window. The components have now been combined into a single function component called First_Last. The input and output parameters have been automatically connected.



Note that inline user-defined functions are displayed with a dashed outline. See Inline user-defined functions for more information.

Dragging the function name from the Libraries pane and dropping it in the mapping window, allows you to use it anywhere in the current mapping. To use it in a different mapping, please see Reusing user-defined functions

To open a user-defined function, do one of the following:

- Double-click the title bar of a user-defined function component
- Double-click the specific user-defined function in the Libraries window.

This displays the individual components inside the function in a tab of that name. Click the Home button to return to the main mapping. Double clicking a user-defined function of a different *.mfd file (in the main mapping window) opens that MFD file in a new tab.

Navigating user-defined functions:

When navigating the various tabs (or user-defined function tabs) in MapForce, a history is automatically generated which allows you to travel forward or backward through the various tabs, by clicking the back/forward icons. The history is session-wide, allowing you to traverse mutliple MFD files.

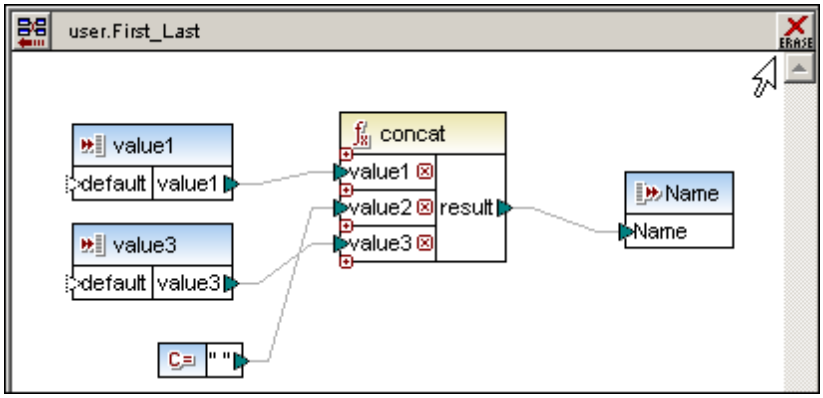
The Home button returns you to the main mapping tab from within the user-defined function.

The Back button takes you back through your history

The Forward button moves you forward through your history

To delete a user-defined function from a library:

1. Double click the specific user-defined function in the Libraries window.
   The user-defined function is visible in its tab.
2. Click the **Erase** button in the top right of the title bar to delete the function.



Reusing - exporting and importing User-defined functions:

User-defined functions defined in one mapping can be imported into any other mapping:

1. Click the **Add/Remove Libraries** button, at the base of the Libraries pane, click the Add button and select a ***.mfd** file that contains the user-defined function(s) you want to import. The user-defined function now appear in the Libraries window (under "user" if that is the default library you selected). You can of course enter anything in the "Library name" field when defining the user-defined function.
2. Drag the imported function into the mapping to make use of it.

Library Names

Note: It is possible to use the same library name for user-defined functions in multiple *.mfd files and/or custom libraries .

Functions from all available sources will appear under the same library name/header in the Libraries pane. However, only the functions in the currently active document can be

 © 2015 Altova GmbH

edited by double-clicking.

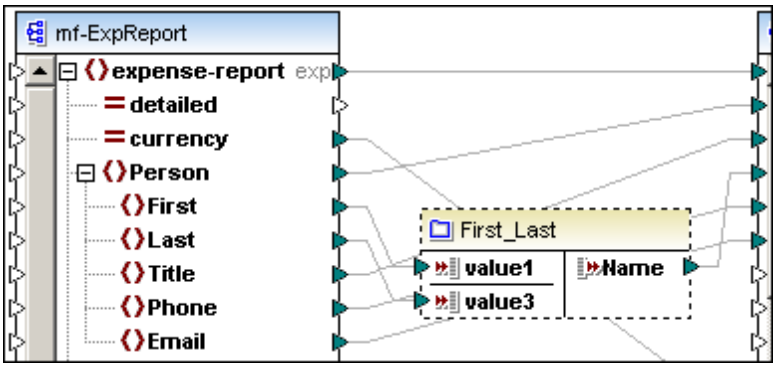
In the following example:

- the function "hello" in the "helloworld" library is imported from a custom,
- the function "Join" in the "helloworld" library is a user-defined function defined in the **current** *.mfd file and
- the function "MyUDF" in the "user" library is also a user-defined function defined in the current *.mfd file

Java Selected

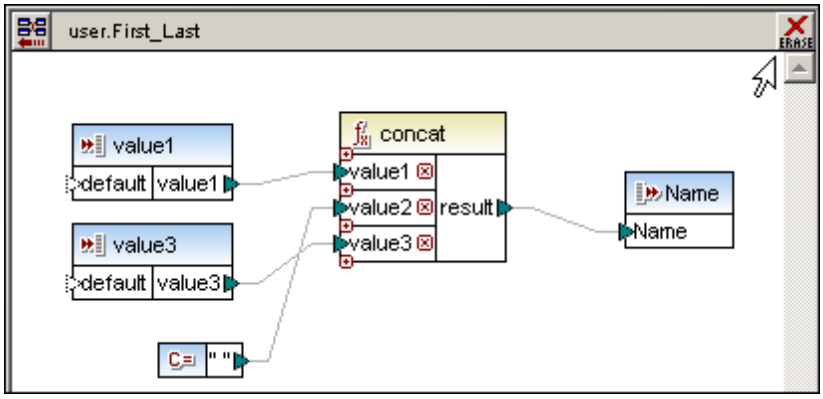
Note that possible changes in imported functions are applied to importing mappings when saving the library ***.mfd** file.

Parameter order in user-defined functions



The parameter order within user-defined functions can be directly influenced:

- Input and output parameters are sorted by their position from top to bottom (from the top left corner of the parameter component).
- If two parameters have the same vertical position, the leftmost takes precedence.
- In the unusual case that two parameters have exactly the same position, the internal component ID is automatically used.



Notes:

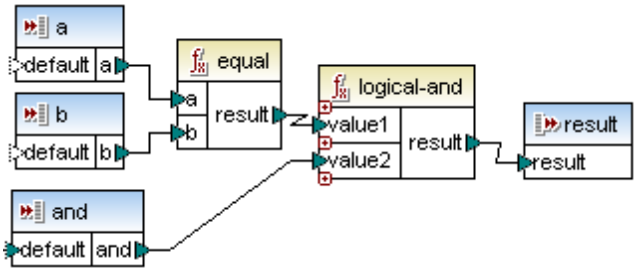
- The Component positioning and resizing actions are undoable.
- Newly added input or output components are created below the last input or output component.
- Complex and simple parameters can be mixed. The parameter order is derived from the component positions.

7.2.1 Function parameters

Function **parameters** are represented inside a user-defined function by **input** and **output components.**

Input components/parameters: **a**, **b**, **and**

Output component/parameter: **result**

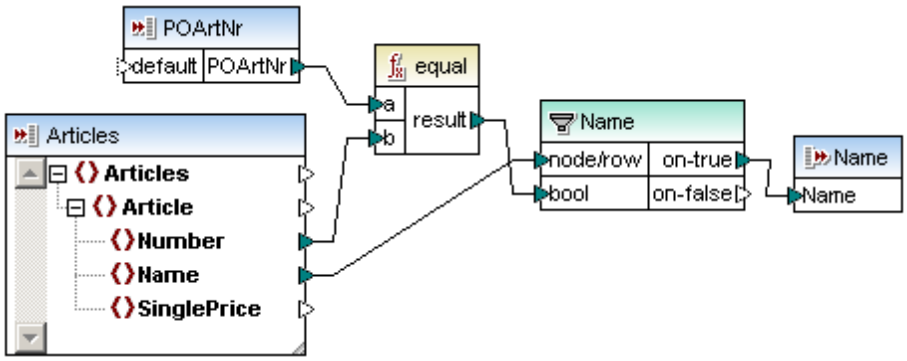


Input parameters are used to pass data from the main mapping into the user-defined function, while output parameters are used to return data back to the main mapping. Note that user-defined functions can also be called from other user-defined functions.

Simple and complex parameters
The input and output parameters of user-defined functions can be of various types:

- Simple values, e.g. string or integer
- Complex node trees, e.g. an XML element with attributes and child nodes

 © 2015 Altova GmbH



Input parameter **POArtNr** is a simple value of datatype "string"

Input parameter **Articles** is a **complex** XML document node tree

Output parameter **Name** is a simple value of type string

Note:

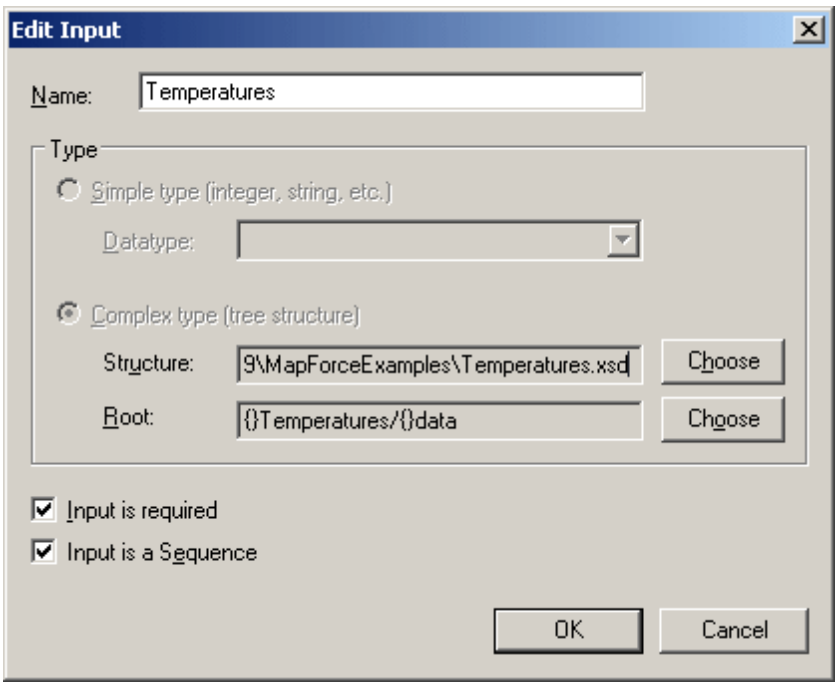
The user-defined functions shown above are all available in the **PersonListByBranchOffice.mfd** file available in the ...\MapForceExamples folder.

Sequences

Sequences are data consisting of a range, or sequence, of values. Simple and complex user-defined **parameters** (input/output) can be defined as sequences in the component properties dialog box.

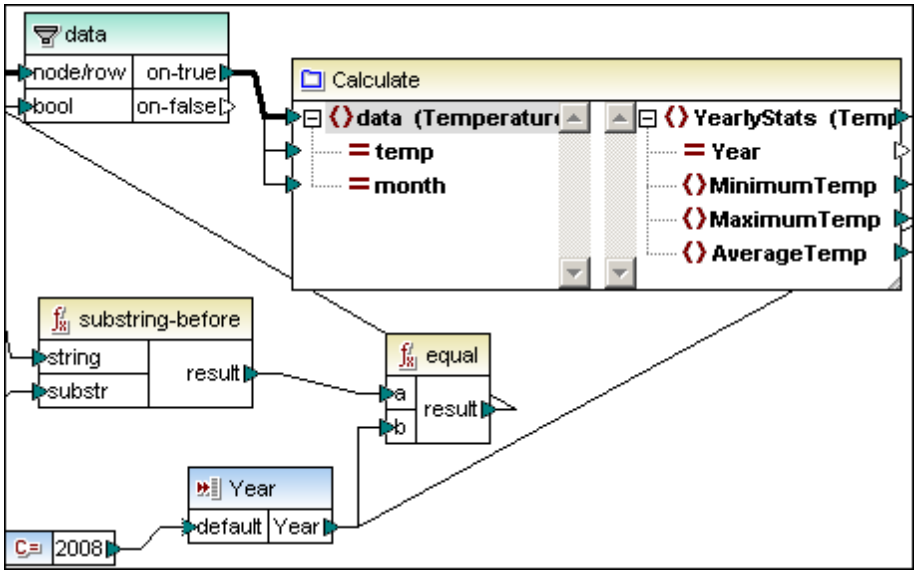
Aggregate functions, e.g. min, max, avg, etc., can use this type of input to supply a single specific value from the input sequence.

When the "**Input is a Sequence**" check box is active, the component handles the input as a sequence. When inactive, input is handled as a single value.

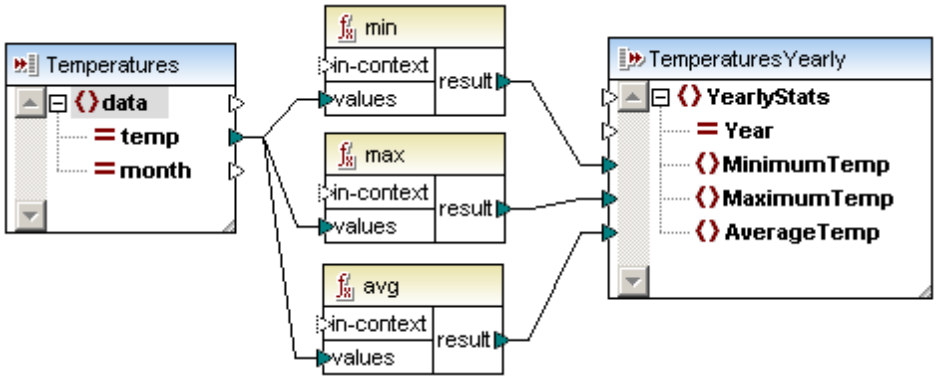


This type of input data, sequence or non-sequence, determines **how often** the function is called.

- When connected to a **sequence** parameter the user-defined function is called only **once** and the complete sequence is passed into the user-defined function.



The screenshot shows the user-defined function "Calculate" of the "**InputIsSequence.mfd**" mapping in the ...\MapForceExamples folder. The **Temperatures** input component (shown below) is defined as a sequence.



- When connected to a **non-sequence** parameter, the user-defined function is called **once** for **each** single item in the sequence.

Please note:

The sequence setting of input/output parameters is ignored when the user-defined function is of type inline.

> Connecting an empty sequence to a non-sequence parameter has the result that the function is not called at all.

This can happen if the source structure has **optional** items, or when a filter condition returns no matching items. To avoid this, either use the substitute-missing function before the function input to ensure that the sequence is never empty, or set the parameter to sequence, and add handling

 © 2015 Altova GmbH

for the empty sequence inside the function.

When a function passes a sequence of multiple values to its output component, and the output component is not set to sequence, only the first result is used when the function is called.

7.2.2 Inline and regular user-defined functions

Inline functions differ fundamentally from regular functions, in the way that they are implemented when code is generated.

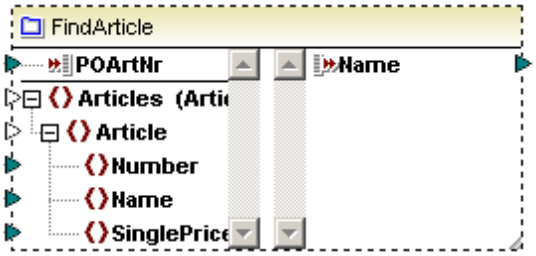
- The code for **inline** type functions is inserted at **all locations** where the user-defined functions are called/used
- The code of a **regular** function is implemented as a function call.

  Inline functions thus behave as if they had been replaced by their implementation. That makes them ideal for breaking down a complex mapping into smaller encapsulated parts.

Please note:

using **inline** functions can significantly increase the amount of generated program code! The user-defined function code is actually inserted at all locations where the function is called/used, and thus increases the code size substantially - as opposed to using a regular function.

INLINE user-defined functions are shown with a **dashed** outline:



Inline user-defined functions **support**:

- **Multiple output components** within a function

do not support:

- The setting of a priority context on a parameter
- Recursive calls to an inline user-defined function

REGULAR user-defined functions i.e. non-inline functions are shown with a **solid** outline:

Regular (non-inline) user-defined functions **support**:

- Only a **single output component** within a function

- **Recursive** calls (where the exit condition must be supplied, e.g. use an If-Else condition where one branch, or value, exits the recursion)
- Setting a priority context on a parameter

Please note:

Although regular functions **do not** support multiple output components, they **can be created** in this type of function. However, an error message appears when you try to generate code, or preview the result of the mapping.

If you are not using recursion in your function, you can change the type of the function to "inline".

do not support:

- Direct connection of filters to simple non-sequence **input** components
- Sequence or aggregate functions on simple input components (like exists, substitute-missing, sum, group-by, ...)

Code generation

The implementation of a regular user-defined function is generated only once as a callable XSLT template or function. Each user-defined function component generates code for a **function call**, where inputs are passed as parameters, and the output is the function (component) return value.

At runtime, all the input parameter values are evaluated first, then the function is called for each occurrence of the input data. See Function parameters for details about this process.

To change the user-defined function "type":

1. Double click the user-defined function to see its constituent components.
2. Select the menu option **Function | Function settings** and click the "Inlined use" checkbox.

User-defined functions and Copy-all connections

When creating Copy-all connections between a schema and a complex user-defined function parameter, the two components must be based on the same schema! It is not necessary that they both have the same root elements however. Please see "Complex output components - defining" for an example.

7.2.3 Creating a simple look-up function

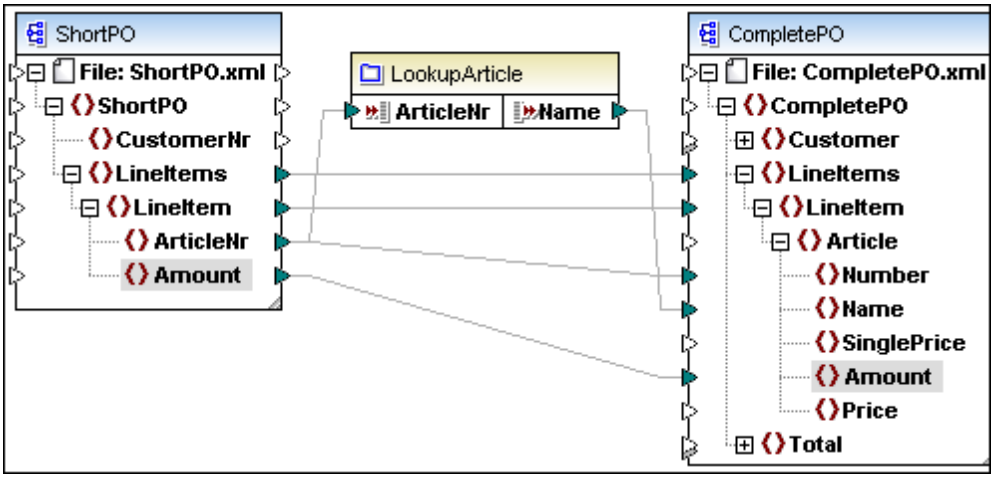
This example is provided as the **lookup-standard.mfd** file available in the **...\MapForceExamples** folder.

Aim:

To create a generic look-up function that:

- supplies Articles/Number data from the Articles XML file, to be compared to Article numbers of a different XML file, ShortPO in this case.

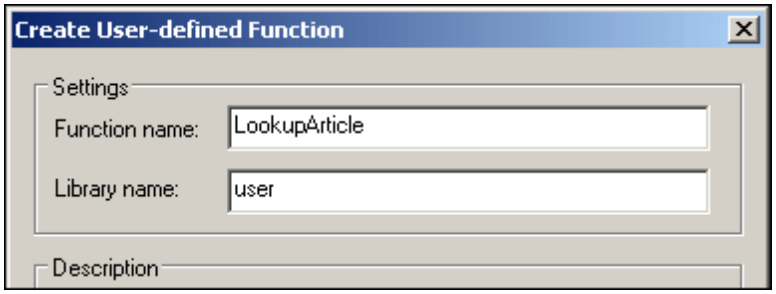
 © 2015 Altova GmbH



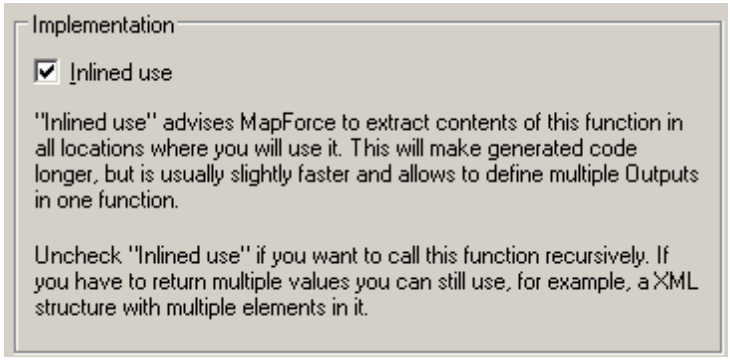
- Insert the ShortPO.xsd and assign ShortPO.xml as the source XML file.
- Insert the CompletePO.xsd schema file, and select CompletePO as the root element.
- Insert a new user-defined function using the method described below.

To create a user-defined function from scratch:

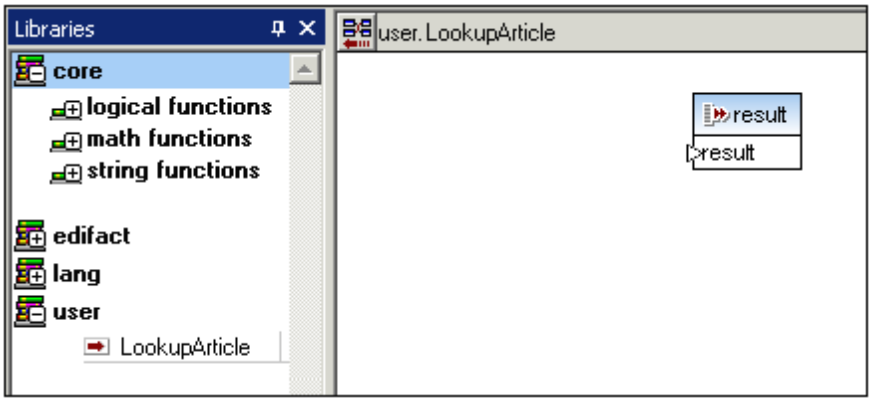
1. Select the menu option **Function | Create User-defined function**.
2. Enter the name of the function e.g. LookupArticle.



3. Uncheck the "Inlined use" checkbox and click OK to confirm


Implementation
Inlined use
"Inlined use" advises MapForce to extract contents of this function in all locations where you will use it. This will make generated code longer, but is usually slightly faster and allows to define multiple Outputs in one function.
Uncheck "Inlined use" if you want to call this function recursively. If you have to return multiple values you can still use, for example, a XML structure with multiple elements in it.

A tab only containing only one item, an output function currently called "result", is displayed.

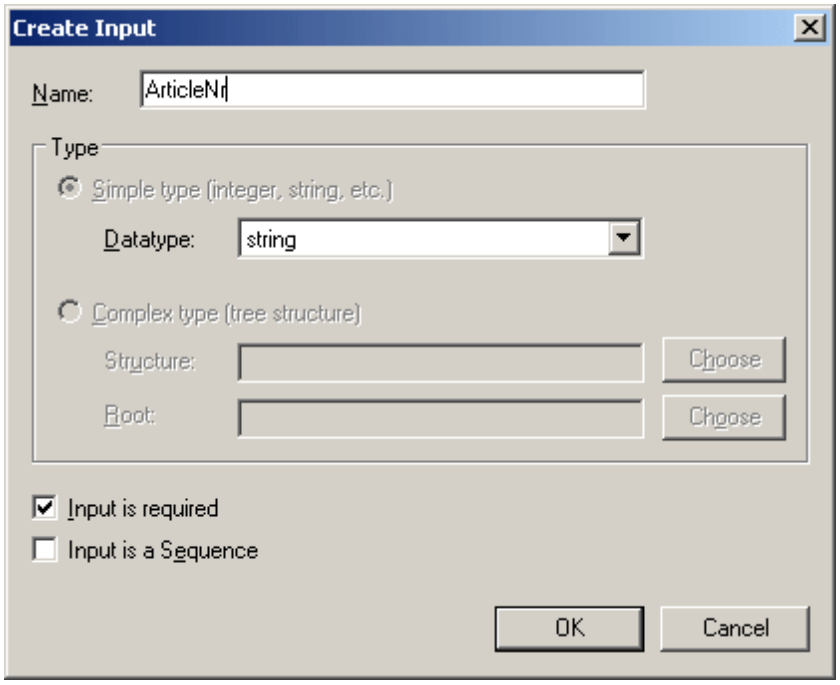


Java Selected

This is the working area used to define the user-defined function.

A new library has been created in the Libraries pane with the name "user" and the function name "LookupArticle".

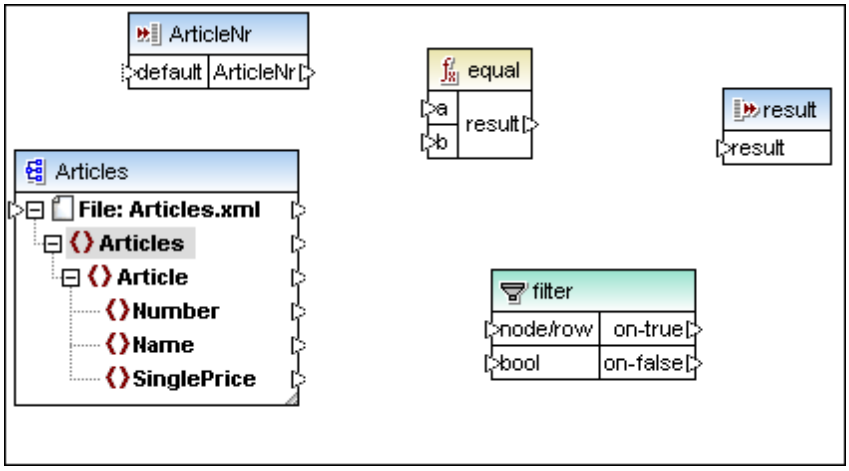
3. Click the **Insert Schema/XML file** icon to insert the **Articles** schema and select the XML file of the same name to act as the data source.
4. Click the **Insert input component** icon to insert an input component.
5. Enter the name of the input parameter, ArticleNr in this case, and click OK.



This component acts as a data input to the user-defined function and supplies the input icon of the user-defined function.

6. Insert an "**equal**" component by dragging it from the core library/logical functions group.
7. Insert a **filter** component by clicking the Insert Filter icon in the toolbar.

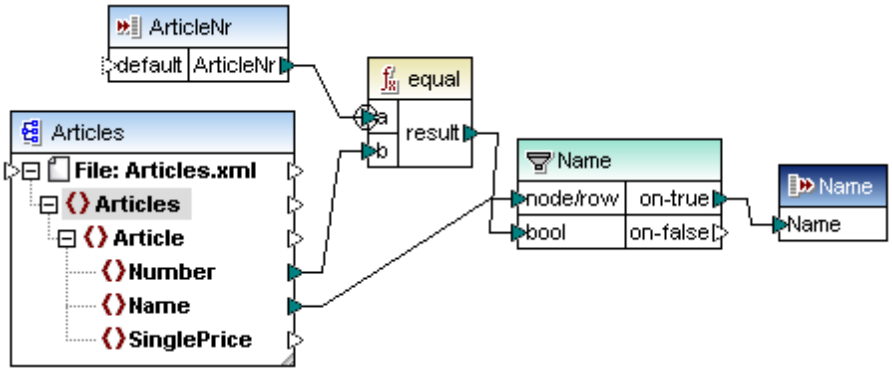
 © 2015 Altova GmbH



Use the diagram below as an aid to creating the mappings in the user-defined function, please take note of the following:

8 Right click the **a** parameter and select **Priority context** from the pop up menu.

9. **Double click** the output function and enter the name of the output parameter, in this case "**Name**".



This ends the definition of the user-defined function.

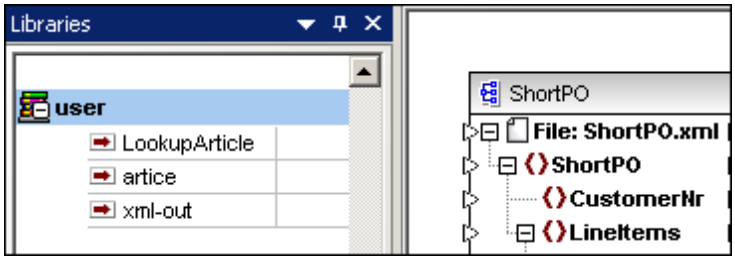
Please note:

Double clicking the **input** and **output** functions opens a dialog box in which you can change the name and datatype of the input parameter, as well as define if the function is to have an input icon (Input is required) and additionally if it should be defined as a sequence.

This user-defined function:

- has one **input** function, ArticleNr, which will receive data from the ShortPO XML file.
- **compares** the ShortPO ArticleNr, with the Article/Number from the **Articles** XML instance file, inserted into the user-defined function for this purpose.
- uses a **filter** component to forward the Article/Name records to the output component, if the comparison returns true.
- has one output function, Name, which will forward the Article Name records to the CompletePO XML file.

10. Click the Home icon to return to the main mapping.
The LookupArticle user-defined function, is now available under the **user** library.

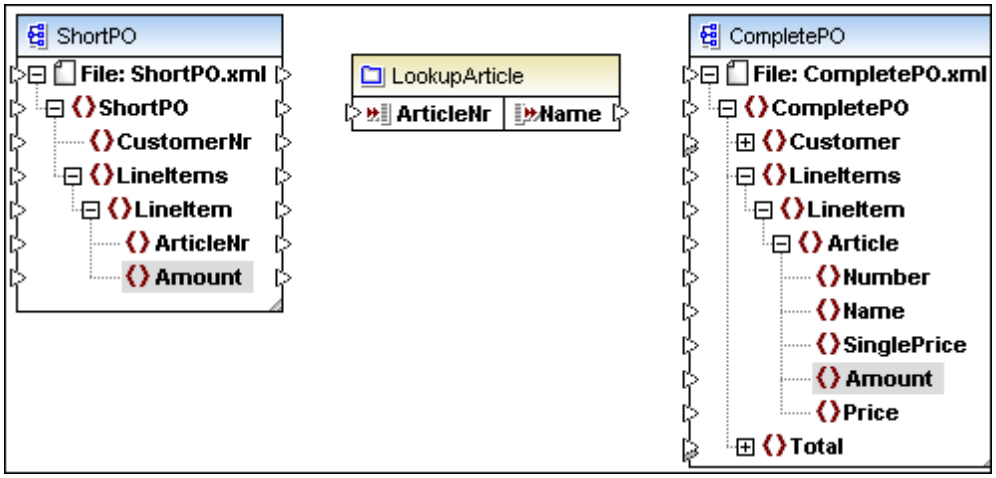


Java Selected

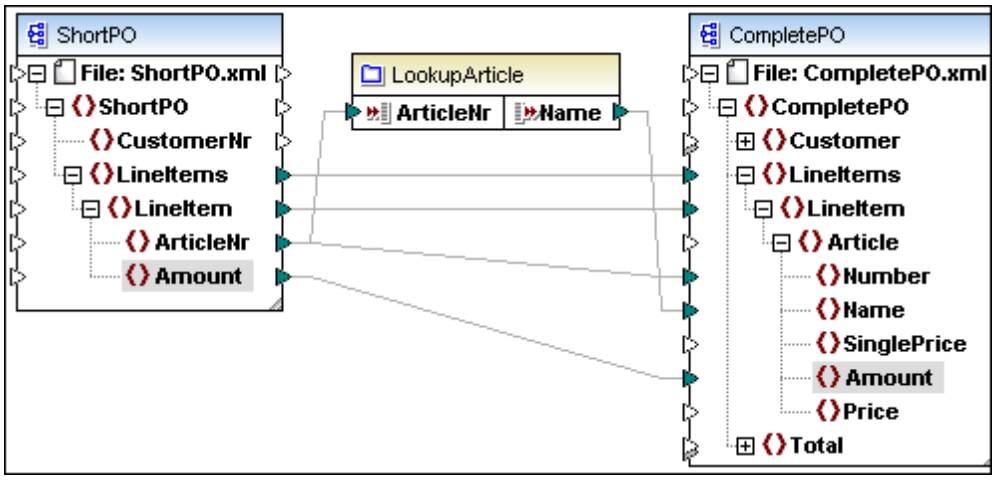
11. Drag the LookupArticle function into the Mapping window.

    The user-defined function is displayed:

- with its name "LookupArticle" in the title/function bar,
- with named input and output icons.



10. Create the connections displayed in the graphic below and click the Output tab to see the result of the mapping.

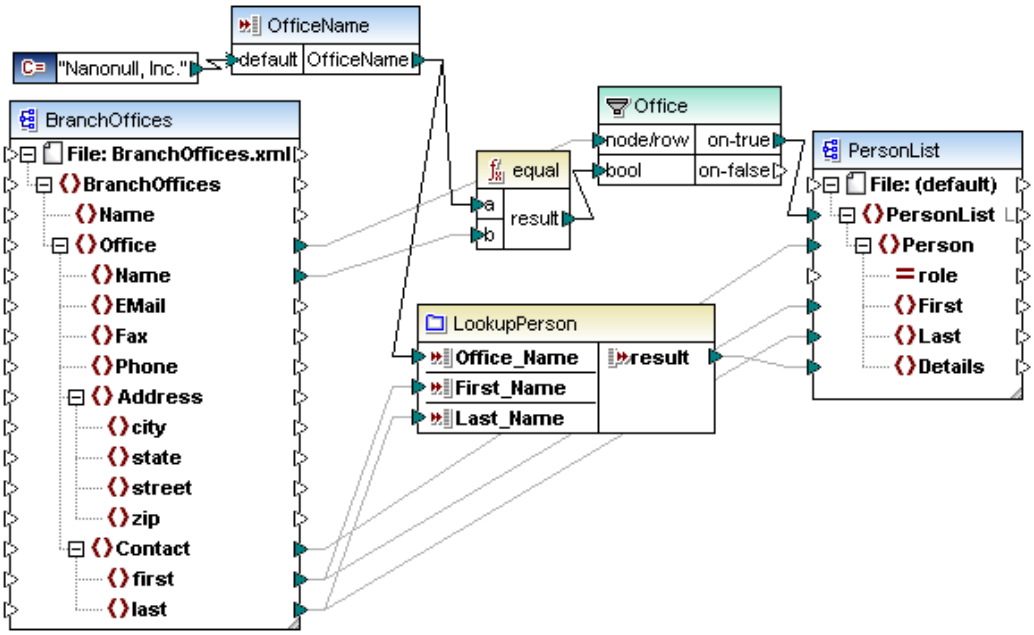


 © 2015 Altova GmbH

7.2.4 User-defined function - example

The **PersonListByBranchOffice.mfd** file available in the **<Documents>\Altova\MapForce2016\MapForceExamples** folder illustrates the following features:

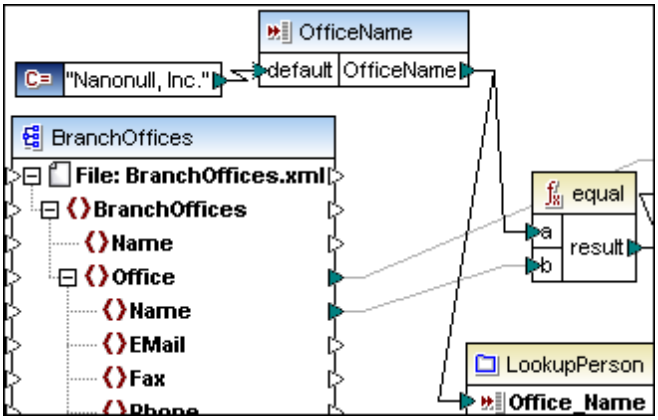
- Nested User-defined functions e.g. **LookupPerson**
- Look-up functions that generate a string output e.g. **LookupPerson**
- **Optional** input-parameters which can also supply a **default** value e.g. the **EqualAnd** component (contained in the LookupPerson component)
- **Configurable** input parameters, which can also double as a command line parameter(s) when executing the generated mapping code!



Configurable input parameters

The input component (OfficeName) receives data supplied when a mapping is executed. This is possible in two ways:

- as a **command line** parameter when executing the generated code, e.g. Mapping.exe / OfficeName "Nanonull Partners, Inc."
- as a **preview** value when using the Built-in execution engine to preview the data in the Output window.

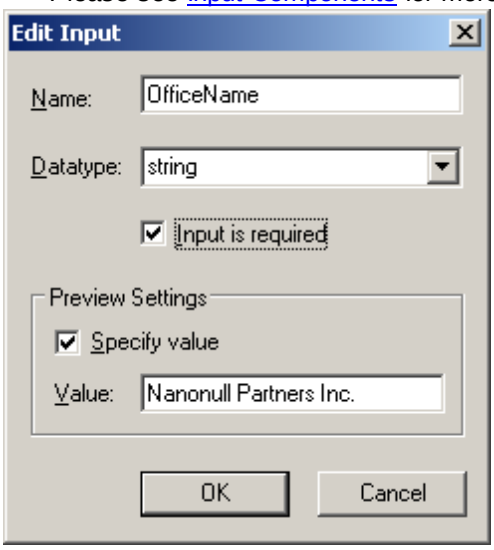


To define the Input value:

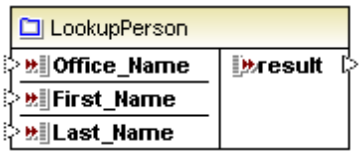
1. Double click the input component and enter a different value in the "Value" text box of the Preview Mode group e.g. "Nanonull Partners, Inc.", and click OK to confirm.
2. Click the Output tab to see the effect.
   A different set of persons are now displayed.

   Please note that the data entered in this dialog box is only used in "**preview**" mode i.e. when clicking the Output tab. If a value is not entered, or the check box is deactivated, then the data mapped to the input icon "default" is used.

   Please see Input Components for more information.



LookupPerson component

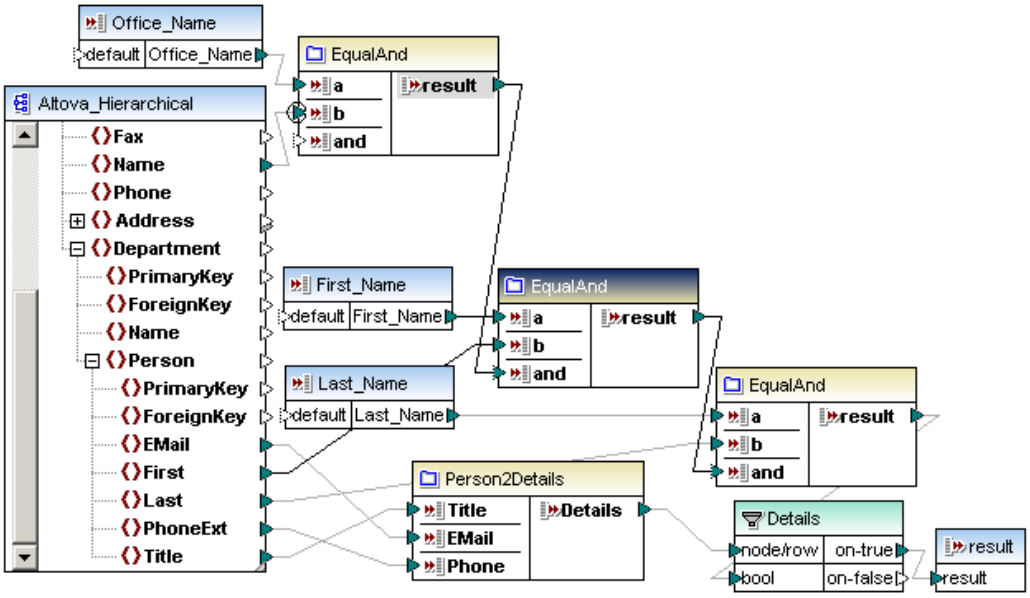


 © 2015 Altova GmbH

Double clicking this user-defined component displays its constituent components shown below. What this component does is:

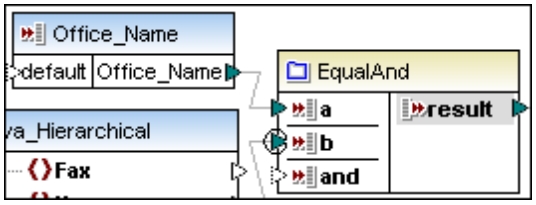
- **Compares** the Office, First, and Last names of BranchOffices.xml, with the same fields of the Altova_Hierarchical.xml file, using the **input** components and the **EqualAnd** user-defined components.
- **Combines** the Email, PhoneExt and Title items using the **Person2Details** user-defined function
- **Passes on** the combined person data to the **output** component if the previous EqualAnd comparisons are all true (i.e. supplied "true" to the filter component).

A user-defined function always outputs a value, which may even be an empty string! This would be the case if the filter component bool value is false. Only an empty string would be output instead of data supplied by the Person2Details component.



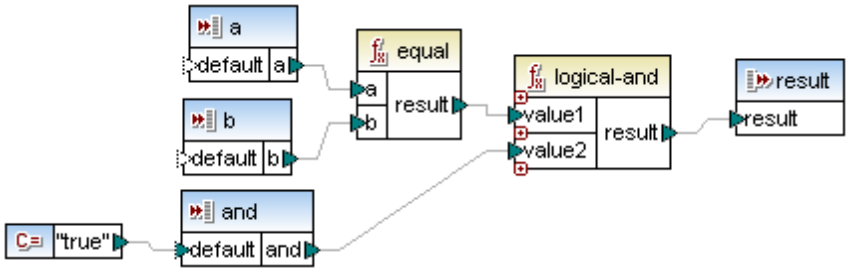
- The three **input** components, Office_Name, First_Name, Last_Name, receive their data from the BranchOffices.xml file.
- The **EqualAnd** component compares two values and provides an **optional** comparison value, as well as a default value.
- Person2Details combines three person fields and passes on the result to the filter component.

EqualAnd component



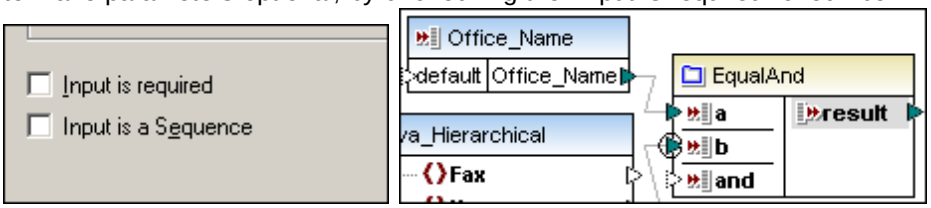
Double clicking this user-defined component displays its constituent components shown below. What this component does is:

- Compare two input parameters **a** and **b**, and pass the result on to the logical-and component. Note that the **b** parameter has been defined as the **priority context** (right click the icon to do so). This ensures that the person data of the specific office, supplied by the input parameter **a**, is processed first.
- **Logical-and** the result of the first comparison, with an **optional** input parameter, "and".
- Pass on the boolean value of this comparison to the output parameter.



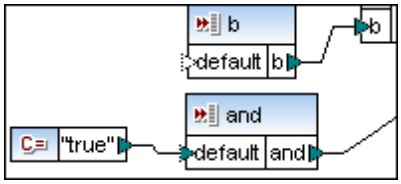
Optional parameters

Double clicking the "and" parameter, of the EqualAnd user-defined function shown above, allows you to make parameters optional, by unchecking the "Input is required" check box.



If "Input is required" is **unchecked**, then:

- A mapping connector is not required for the input icon of this user-defined function, e.g. the **and** parameter of the first EqualAnd function, does not have an input connector. The input icon has a dashed outline to show this visually.
- A **default** value can be supplied by connecting a component, within the user-defined function e.g. using a constant component containing the value "true".



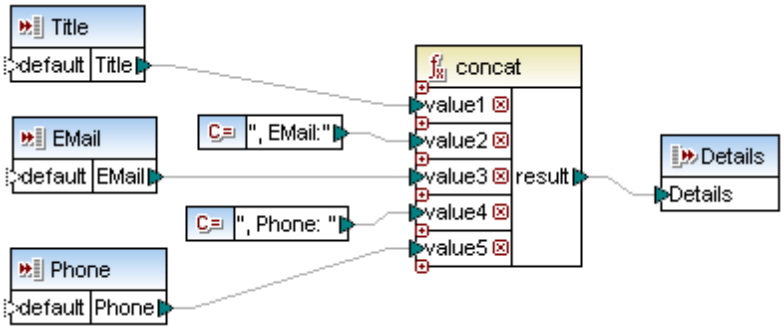
- A mapping from another item, mapped to the optional Input, takes precedence over the default value. E.g. the "and" parameter of second EqualAnd function, receives input data from the "result" parameter of the first EqualAnd user-defined function.

 © 2015 Altova GmbH

Person2Details component

Double clicking this user-defined component displays its constituent components shown below. What this component does is:

- Concatenate three inputs and pass on the result string to the output parameter.
- Double clicking an output parameter allows you to change the parameter name (Details), and select the datatype (String).



7.2.5 Complex user-defined function - XML node as input

This example is provided as the **lookup-udf-in.mfd** file available in the ...\MapForceExamples folder. What this section will show, is how to define an inline user-defined function that contains a complex input component.

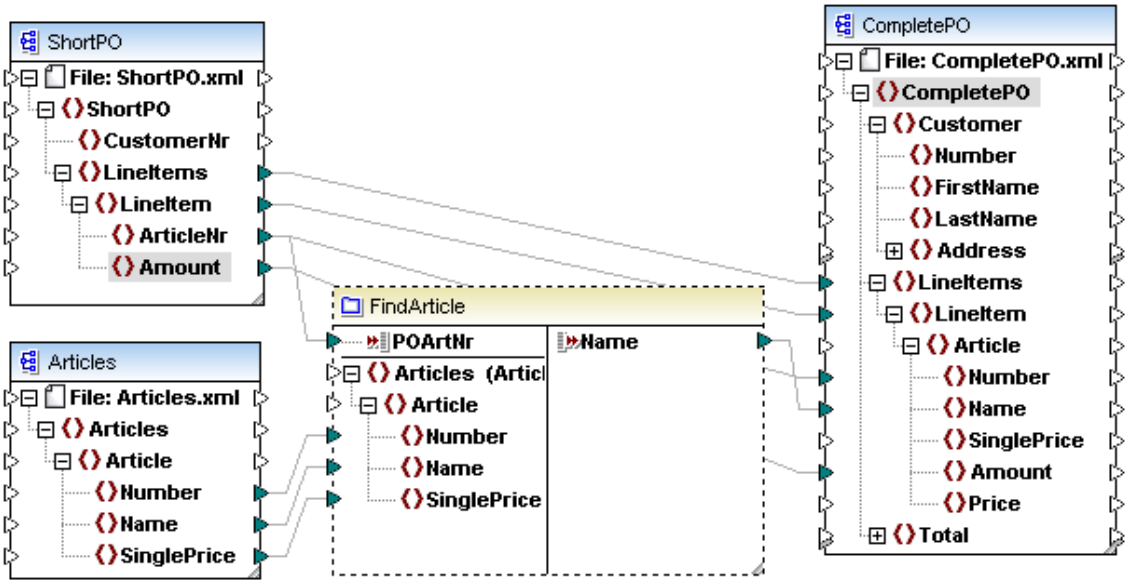
Note that the user-defined function "FindArticle" consists of two halves.

A left half which contains the input parameters:

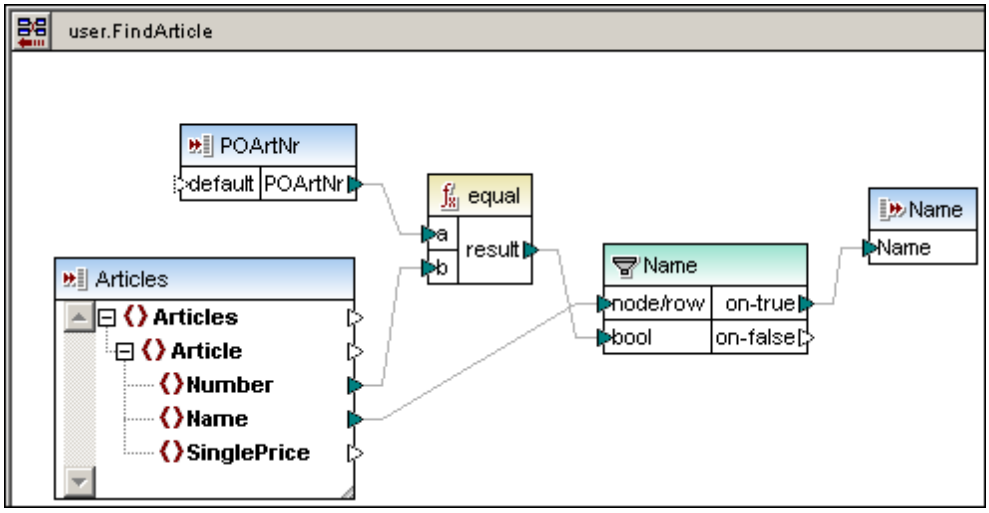
- a simple input parameter **POArtNr**
- a complex input component **Articles,** with mappings directly to its XML child nodes

A right half which contains:

- a simple output parameter called "**Name"**.



The screenshot below shows the constituent components of the user-defined function, the two input components at left and the output component at right.

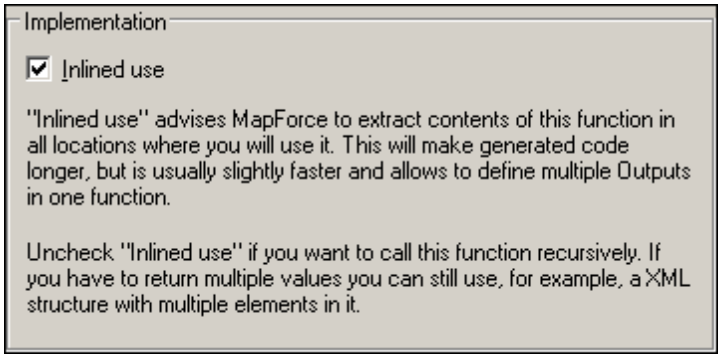


7.2.5.1 *Defining Complex Input Components*

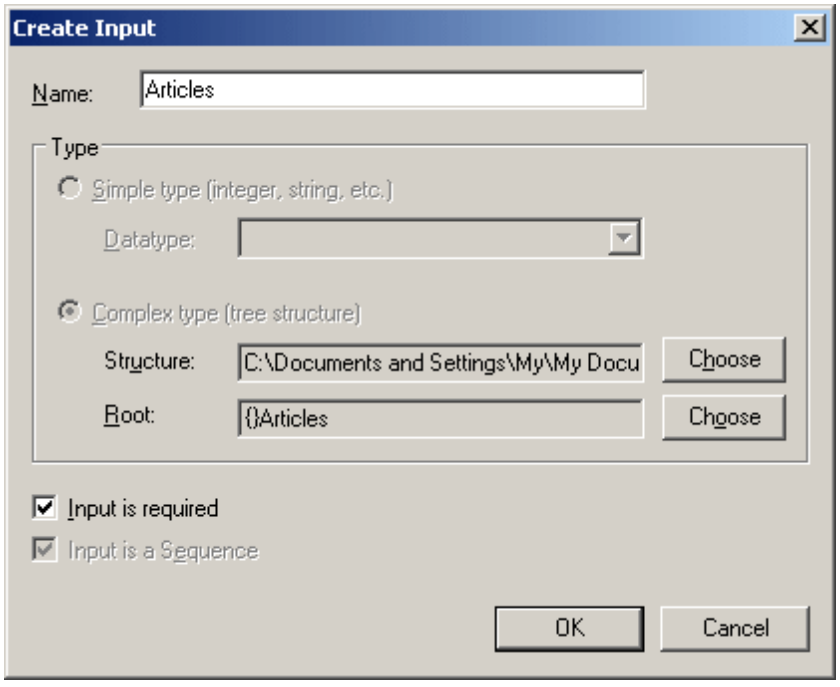
Defining complex input components:

1. Create a user-defined function in the usual manner, i.e. **Function | Create User-Defined function** and click OK to confirm. Note that the **Inlined use** check box is automatically selected.

 © 2015 Altova GmbH


Implementation
Inlined use
"Inlined use" advises MapForce to extract contents of this function in all locations where you will use it. This will make generated code longer, but is usually slightly faster and allows to define multiple Outputs in one function.
Uncheck "Inlined use" if you want to call this function recursively. If you have to return multiple values you can still use, for example, a XML structure with multiple elements in it.

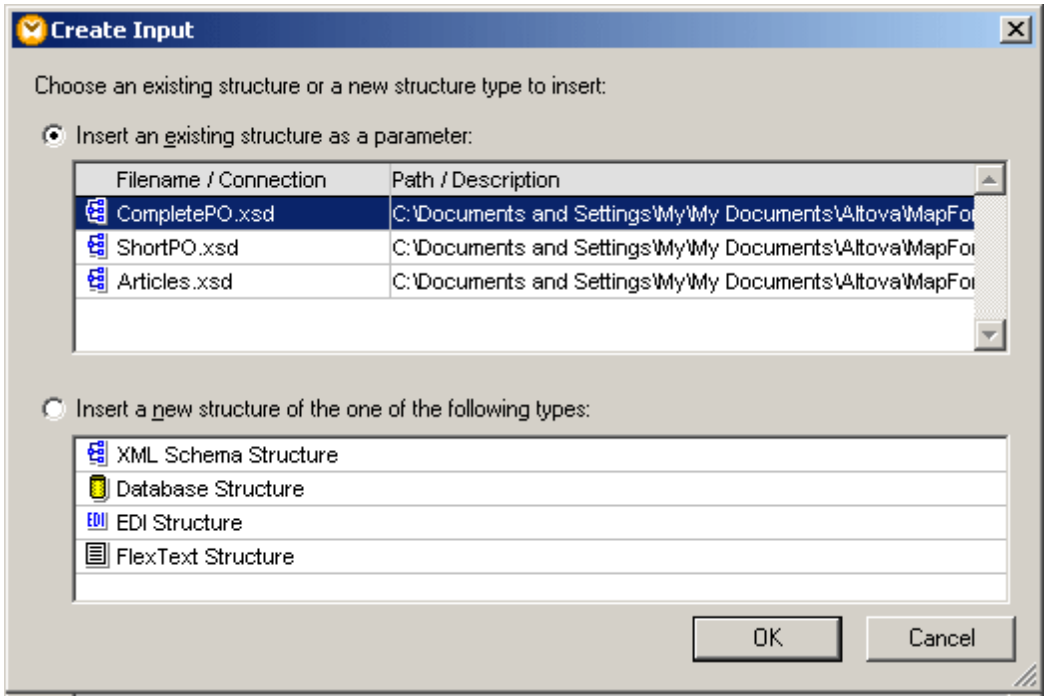
2. Click the **Insert input component** icon in the icon bar.
3. Enter the name of the input component into the Name field.



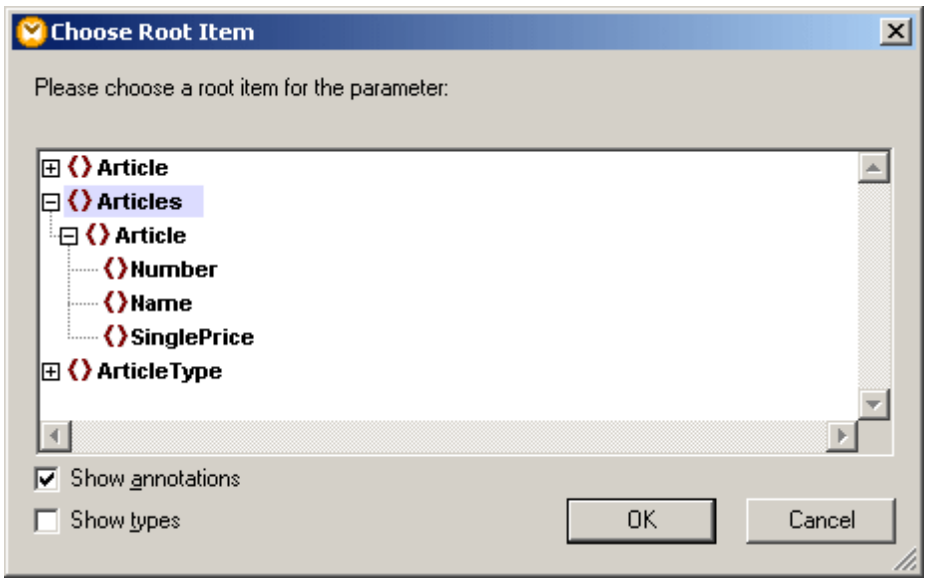
4. Click the **Complex type (tree structure)** radio button, then click the "**Choose**" button next to the Structure field.
   This opens another dialog box.

   The top list box displays the **existing** components in the mapping (three schemas if you opened the example mapping). Note that this list contains all of the components that have been inserted into the active mapping: e.g. XML schema file.

   The lower list box allows you to select a new complex data structure i.e. XML Schema file.

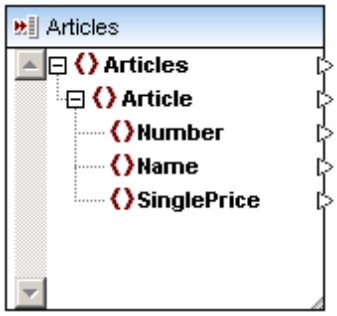


5. Click "Insert a new structure... " radio button, select the XML Schema Structure entry, and click OK to continue.
6. Select **Articles.xsd** from the "Open" dialog box.
7. Click the element that you would like to become the root element in the component, e.g. Articles, and click OK, then OK again to close both dialog boxes.

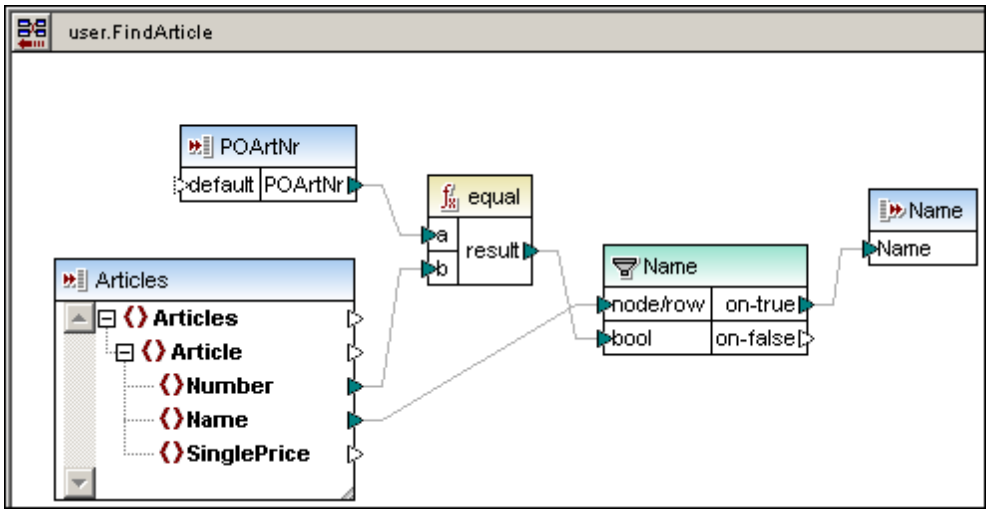


The Articles component is inserted into the user-defined function. Please note the input icon to the left of the component name. This shows that the component is used as a complex input component.

 © 2015 Altova GmbH



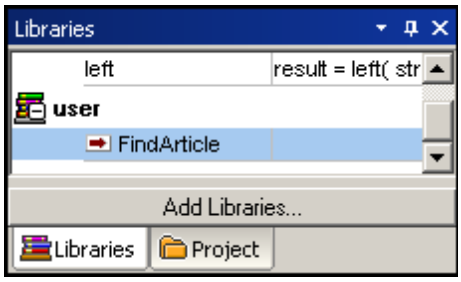
8. Insert the rest of the components as shown in the screenshot below, namely: a second "simple" input component (called POArtNr), filter, equal and output component (called Name), and connect them as shown.



Please note:

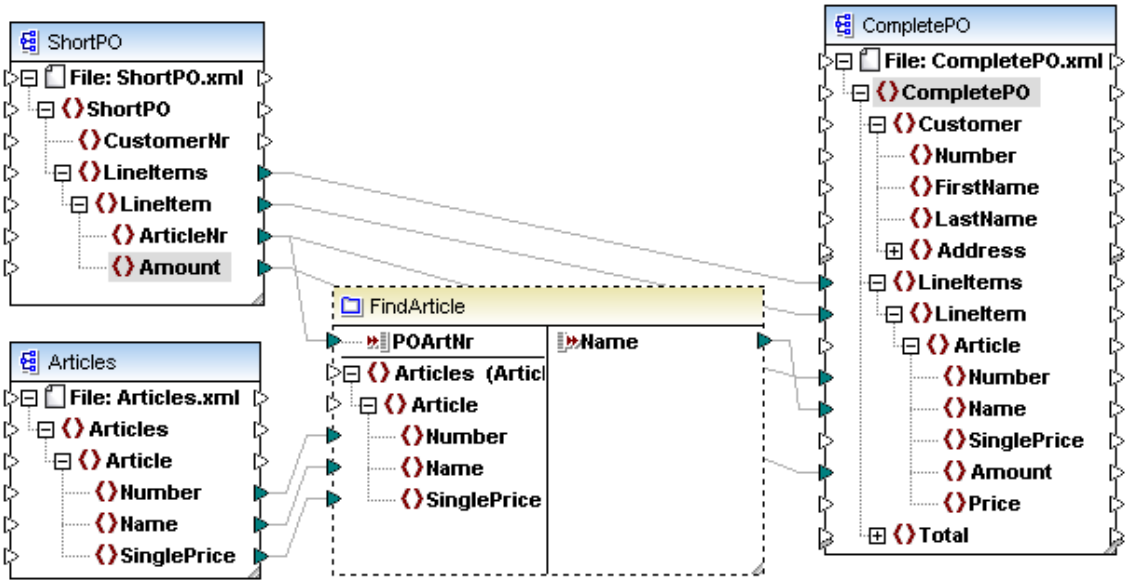
- The **Articles** input component receives its data from **outside** of the user-defined function. Input icons that allow mapping to this component, are available there.
- An XML **instance** file to provide data from within the user-defined function, cannot be assigned to a complex input component.
- The other input component POArtNr, supplies the ShortPO article number data to which the **Article | Number** is compared.
- The filter component filters the records where both numbers are identical, and passes them on to the output component.

10. Click the Home icon to return to the mapping.
11. Drag the newly created user-defined component from the Libraries pane into the mapping.



Java Selected

12. Create the connections as shown in the screenshot below.



The left half contains the input parameters to which items from two schema/xml files are mapped:

- **ShortPO** supplies the data for the input component **POArtNr.**
- **Articles** supplies the data for the complex input component. The Articles.xml instance file was assigned to the Articles schema file when the component was inserted.
- The complex input component **Articles** with its XML child nodes, to which data has been mapped from the Articles component.

The right half contains:

- a simple output parameter called "**Name"**, which passes on the filtered line items which have the same Article number, to the Name item of Complete PO.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <CompletePO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    <LineItems>
4      <LineItem>
5        <Article>
6          <Number>3</Number>
7          <Name>Pants</Name>
8          <Amount>5</Amount>
9        </Article>
10     </LineItem>
11     <LineItem>
12       <Article>
13         <Number>1</Number>
14         <Name>T-Shirt</Name>
15         <Amount>17</Amount>
16       </Article>
17     </LineItem>
18   </LineItems>
19 </CompletePO>
```

Please note:
When creating **Copy-all** connections between a schema and a user-defined function parameter, the two components must be based on the same schema! It is not necessary that they both have the same root elements however.

 © 2015 Altova GmbH

7.2.6 Complex user-defined function - XML node as output

This example is provided as the **lookup-udf-out.mfd** file available in the **...\MapForceExamples** folder. What this section will show is how to define an inline user-defined function that allows a complex output component.

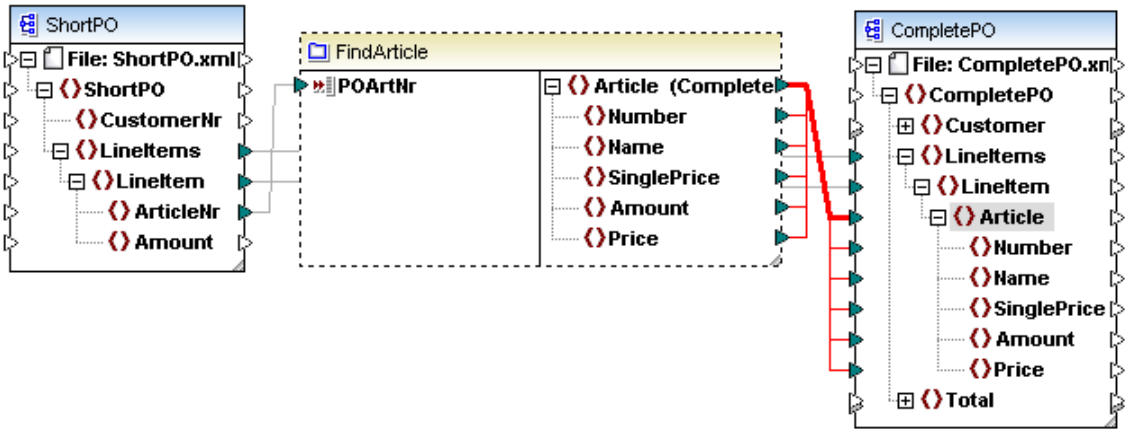
Note that the user-defined function FindArticle consists of two halves.

A left half which contains the input parameter:

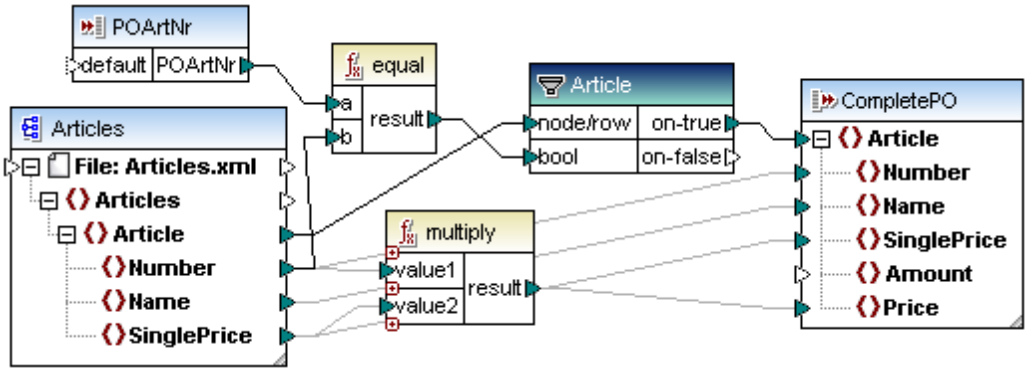
- a simple input parameter **POArtNr**

A right half which contains:

- a complex output component **Article (CompletePO)** with its XML child nodes mapped to CompletePO.



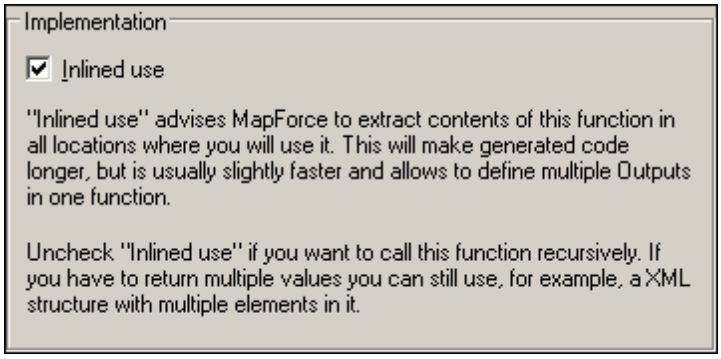
The screenshot below shows the constituent components of the user-defined function, the input components at left and the complex output component at right.



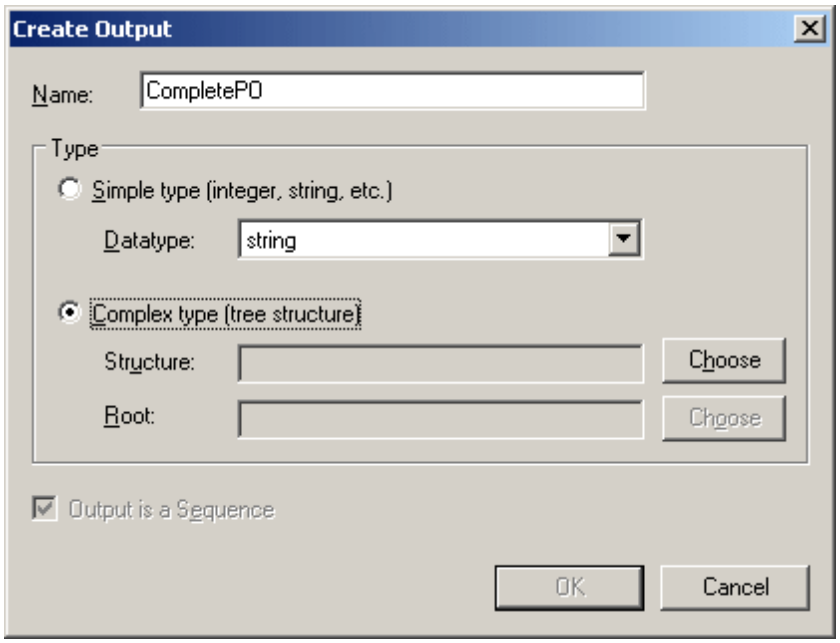
7.2.6.1 *Defining Complex Output Components*

Defining complex output components:

1. Create a user-defined function in the usual manner, i.e. **Function | Create User-Defined function** name it **FindArticle**, and click OK to confirm. Note that the **Inline...** option is automatically selected.


Implementation
Inlined use
"Inlined use" advises MapForce to extract contents of this function in all locations where you will use it. This will make generated code longer, but is usually slightly faster and allows to define multiple Outputs in one function.
Uncheck "Inlined use" if you want to call this function recursively. If you have to return multiple values you can still use, for example, a XML structure with multiple elements in it.

2. Click the Insert **Output** icon in the icon bar, and enter a name e.g. CompletePO.

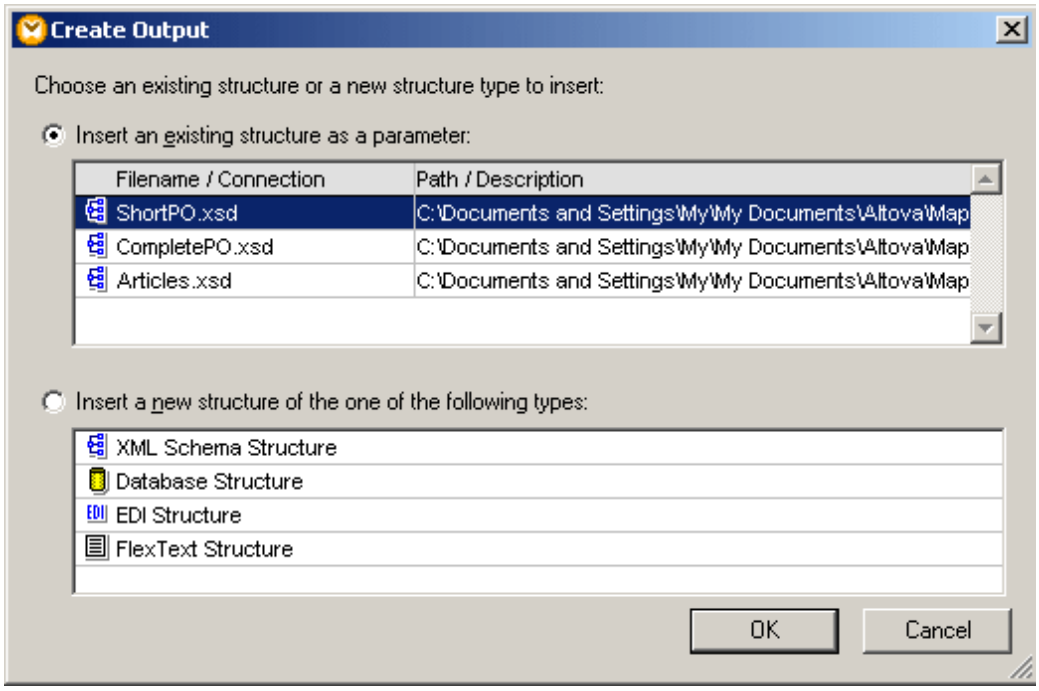


3. Click the **Complex type...** radio button, then click the "**Choose**" button.
This opens another dialog box.

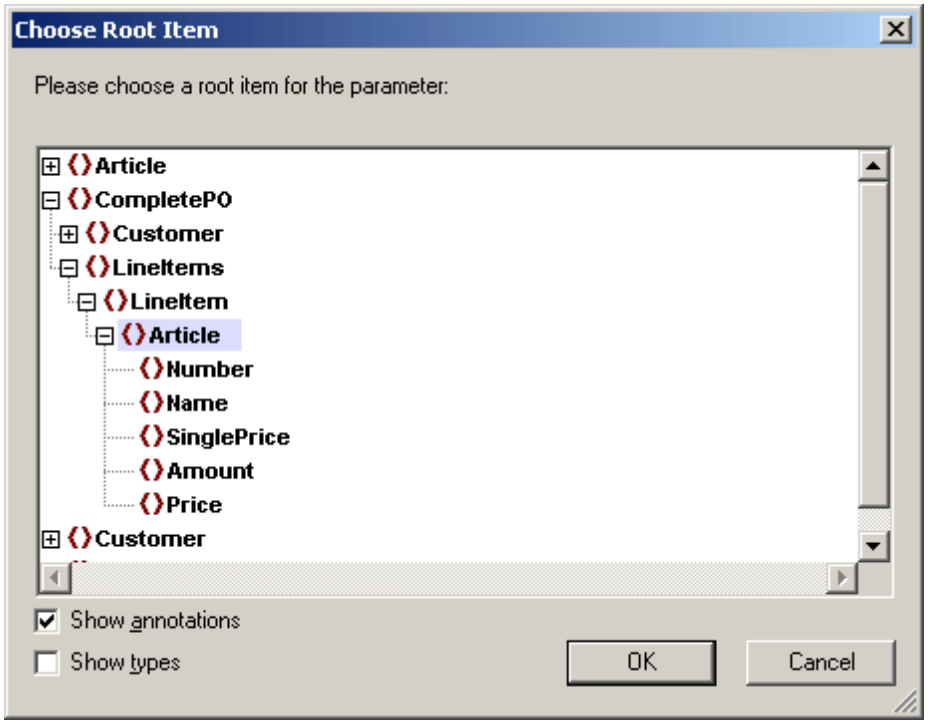
   The top list box displays the **existing** components in the mapping, (three schemas if you opened the example file). Note that this list contains all of the components that have been inserted into the active mapping: e.g. XML Schema file.

   The lower list box allows you to select a new complex data structure i.e. XML Schema file.

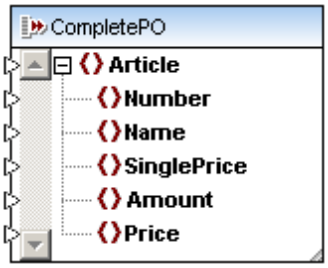
 © 2015 Altova GmbH



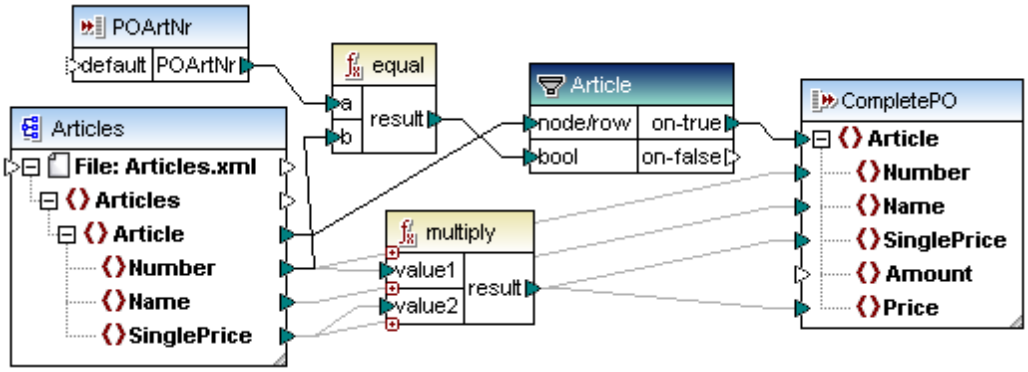
4. Click "**Insert new structure...** " radio button, select the XML Schema Structure entry, and click OK to continue.
5. Select the **CompletePO.xsd** from the "Open" dialog box.
6. Click the element that you would like to become the root element in the component, e.g. **Article**, and click OK, then OK again to close the dialog boxes.



The CompletePO component is inserted into the user-defined function. Please note the output icon to the left of the component name. This shows that the component is used as a complex output component.



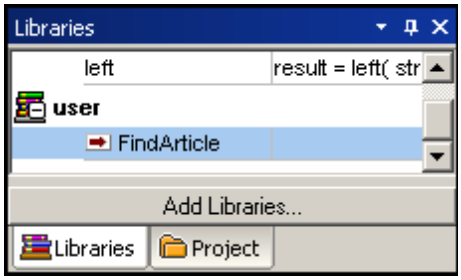
7. Insert the Articles schema/XML file into the user-defined function and assign the **Articles.xml** as the XML instance.
8. Insert the rest of the components as shown in the screenshot below, namely: the "simple" input components (POArtNr), filter, equal and multiply components, and connect them as shown.



Please note:

- The **Articles** component receives its data from the Articles.xml instance file, within the user-defined function.
- The input components supply the POArtNr (article number) and Amount data to which the Articles | Number & Price are compared.
- The filter component filters the records where both numbers are identical, and passes them on to the CompletePO output component.

9. Click the Home icon to return to the mapping.
10. Drag the newly created user-defined component from the Libraries pane into the mapping.

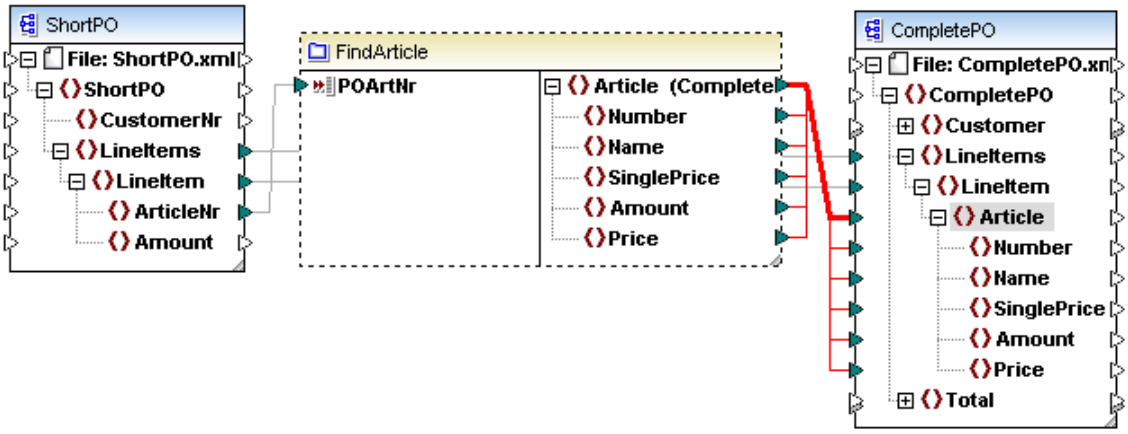


Java Selected

11. Create the connections as shown in the screenshot below.
Having created the Article (CompletePO) connector to the target, right click it and select "Copy-all" from the context menu. The rest of the connectors are automatically

 © 2015 Altova GmbH

generated, and are highlighted in the screenshot below.



Please note:
When creating **Copy-all** connections between a schema and a user-defined function of type "Inline", the two components must be based on the same schema! It is not necessary that they both have the same root elements however.

The left half contains the input parameter to which a single item is mapped:

- ShortPO supplies the article number to the **POArtNr** input component.

The right half contains:

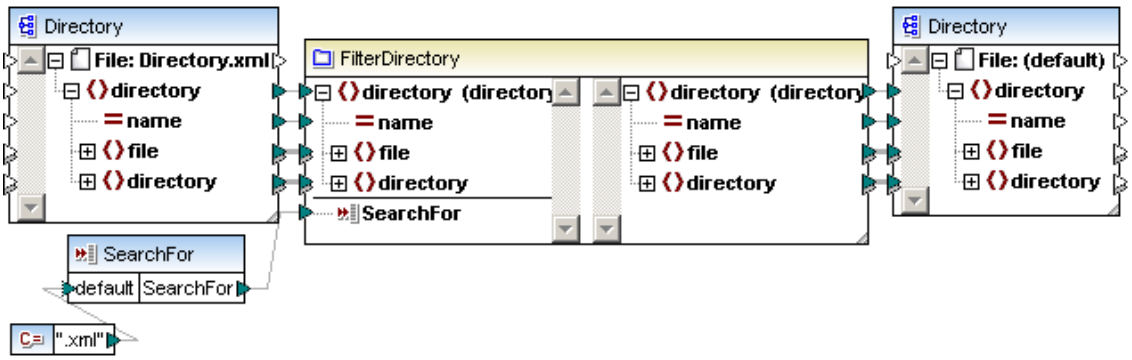
- a complex output component called "**Article (CompletePO)**" with its XML child nodes, which maps the filtered items, of the same Article number, to CompletePO.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <CompletePO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespace
3    <LineItems>
4      <LineItem>
5        <Article>
6          <Number>3</Number>
7          <Name>Pants</Name>
8          <SinglePrice>34</SinglePrice>
9          <Price>102</Price>
10       </Article>
11     </LineItem>
12     <LineItem>
13       <Article>
14         <Number>1</Number>
15         <Name>T-Shirt</Name>
16         <SinglePrice>25</SinglePrice>
17         <Price>25</Price>
18       </Article>
```

7.2.7 Recursive user-defined mapping

This section will describe how the mapping **RecursiveDirectoryFilter.mfd**, available in the ... **\MapForceExamples** folder, was created and how recursive mappings are designed. The MapForceExamples project folder contains further examples of recursive mappings.

The screenshot below shows the finished mapping containing the recursive user-defined function FilterDirectory, the aim being to filter a list of the .xml files in the source file.

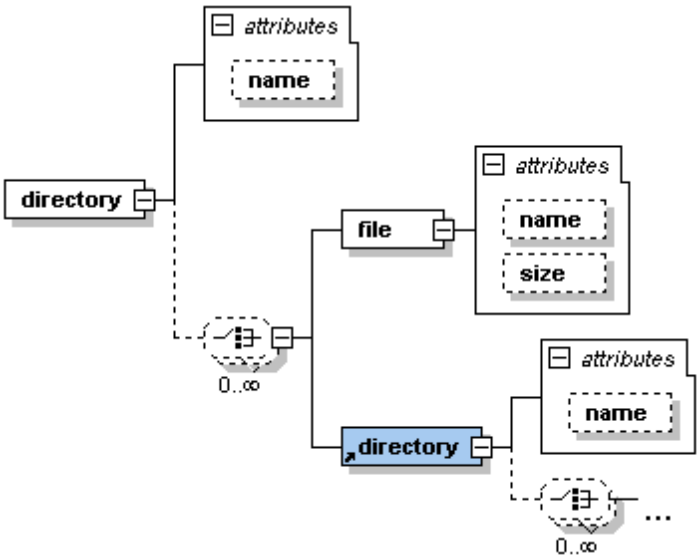


The **source file** that contains the file and directory data for this mapping is Directory.xml. This XML file supplies the directory and file data in the hierarchical form you see below.

```
24      <directory name="output">
25        <file name="examplesite1.css" size="3174"/>
26        <directory name="images">
27          <file name="blank.gif" size="88"/>
28          <file name="block_file.gif" size="13179"/>
29          <file name="block_schema.gif" size="9211"/>
30          <file name="nav_file.gif" size="60868"/>
31          <file name="nav_schema.gif" size="6002"/>
32        </directory>
33      </directory>
34    </directory>
35    <directory name="Import">
36      <file name="altova.mdb" size="266240"/>
37      <file name="Data_shape.mdb" size="225280"/>
38    </directory>
39    <directory name="IndustryStandards">
40      <directory name="News">
41        <file name="high-tide.jpg" size="10793"/>
42        <file name="Newsml-example.xml" size="5004"/>
43        <file name="nitf-example.xml" size="9327"/>
44      </directory>
```

The XML schema file referenced by Directory.xml has a **recursive** element called "directory" which allows for any number of subdirectories and files below the directory element.

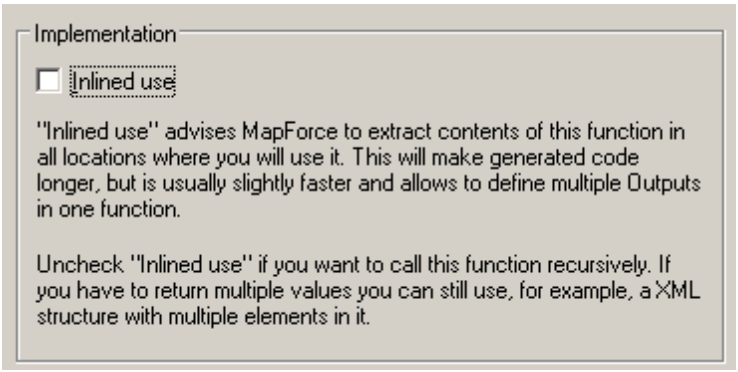
 © 2015 Altova GmbH



7.2.7.1 Defining a recursive user-defined function

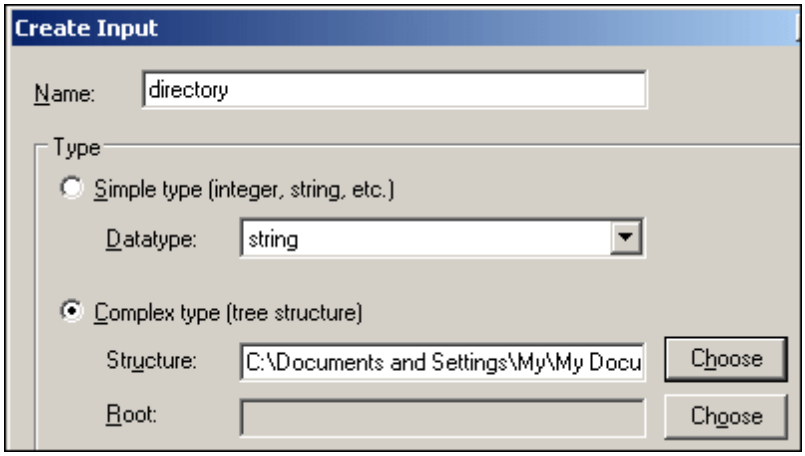
From the **main** mapping window:

1. Select **Function | Create User defined Function** to start designing the function and enter a name e.g. FilterDirectory.
2. Make sure that you **deselect** the **Inlined Use** check box in the Implementation group, to make the function recursive, then click OK.

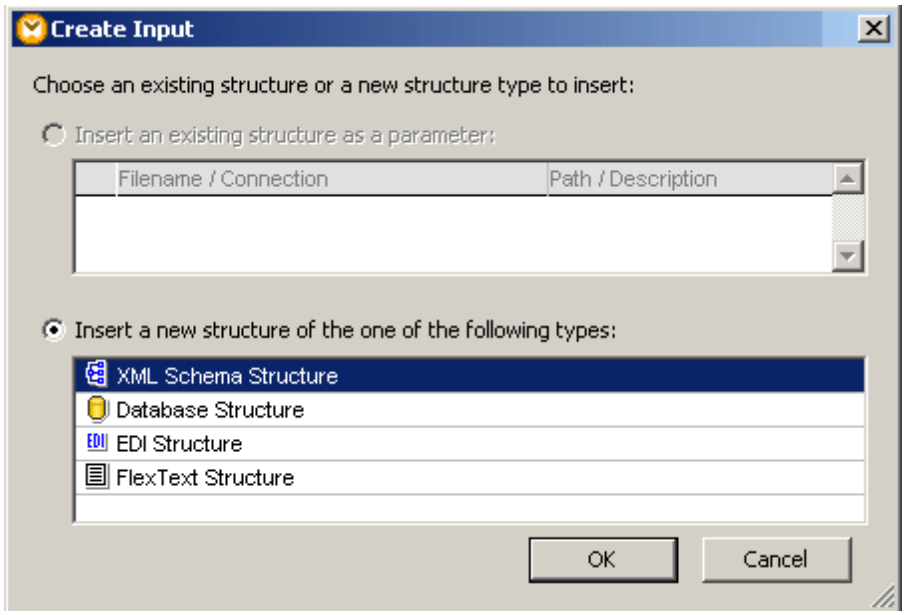

Implementation
Inlined use
"Inlined use" advises MapForce to extract contents of this function in all locations where you will use it. This will make generated code longer, but is usually slightly faster and allows to define multiple Outputs in one function.
Uncheck "Inlined use" if you want to call this function recursively. If you have to return multiple values you can still use, for example, a XML structure with multiple elements in it.

You are now in the **FilterDirectory** window where you create the user-defined function.

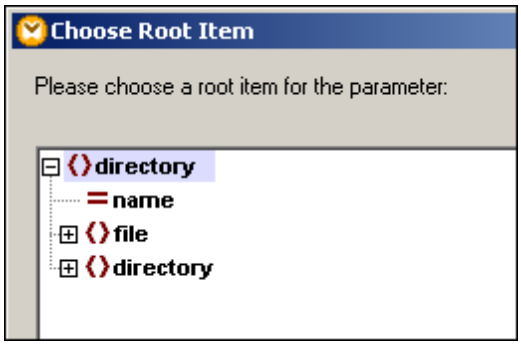
3. Select **Function | Insert Input** to insert an input component.
4. Give the component a name e.g. "directory" and click on the **Complex Type** (tree structure) radio button.



5. Click the **Choose** button, click the "XML Schema Structure" entry in the lower pane, then click OK.


Create Input
Choose an existing structure or a new structure type to insert:
Insert an existing structure as a parameter:
Filename / Connection
Path / Description
Insert a new structure of the one of the following types:
XML Schema Structure
Database Structure
EDI Structure
FlexText Structure
OK
Cancel

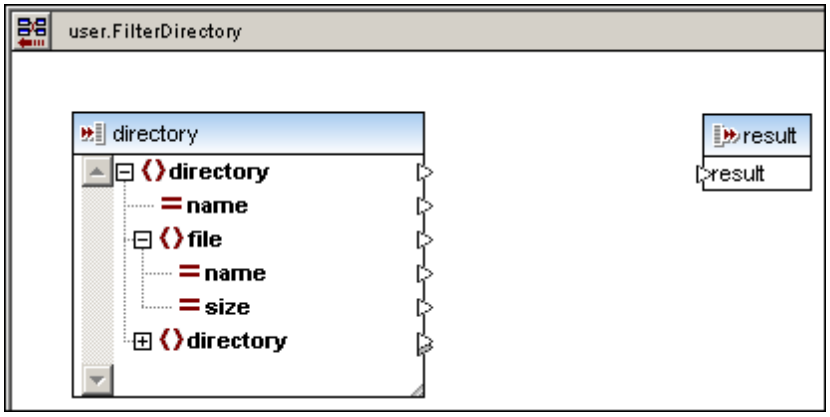
6. Select the **Directory.xsd** file in the ...\MapForceExamples folder and click the Open button.
7. Click OK again when asked to select the root item, which should be "directory" as shown below.



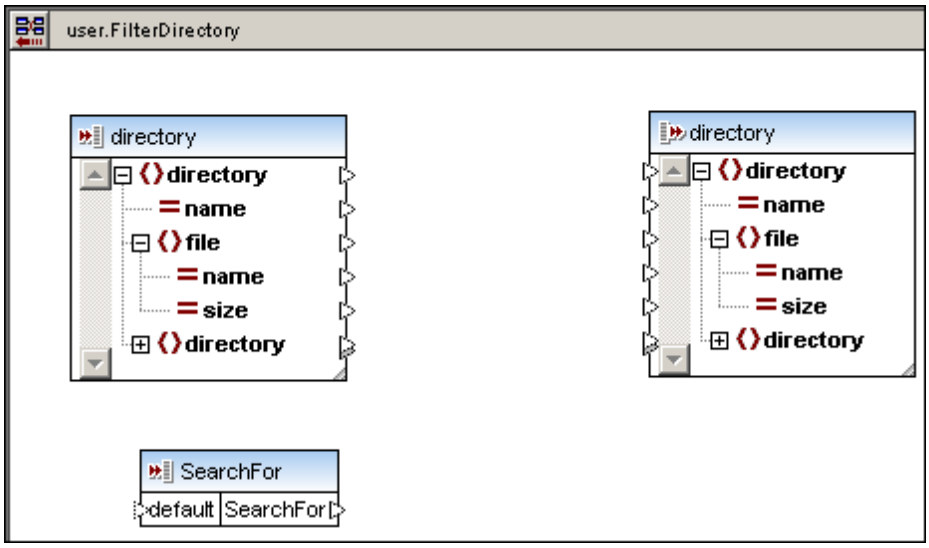
8. Click OK again to insert the complex input parameter.

 © 2015 Altova GmbH

The user-defined function is shown below.



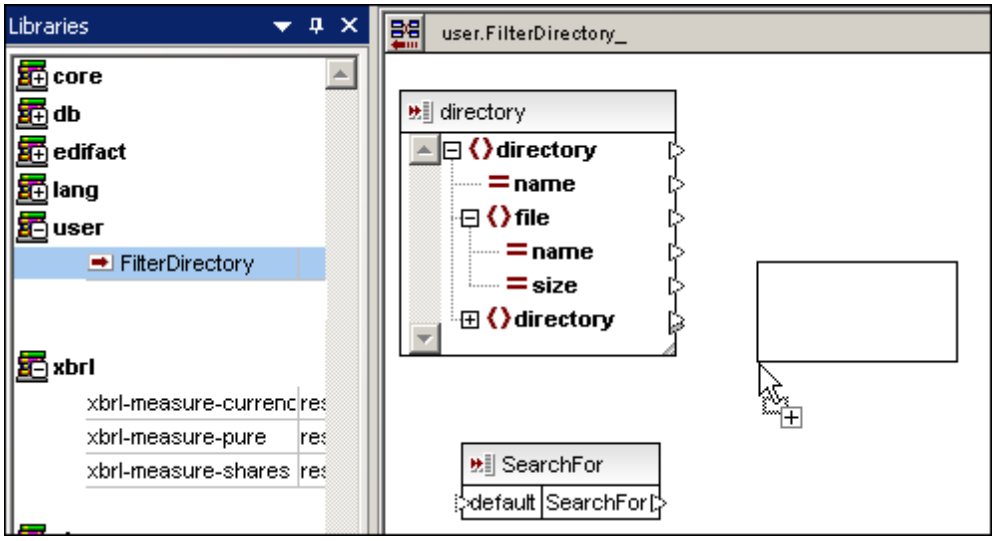
9. Delete the simple result output component, as we need to insert a complex output component here.
10. Select **Function | Insert Output...** to insert an **output** component and use the same method as outlined above, to make the output component, "directory", a complex type. You now have two complex components, one input and the other output.
11. Select **Function | Insert Input...** and insert a component of type Simple type, and enter a name e.g. **SearchFor**. Deselect the "Input is required" checkbox.



Inserting the recursive user-defined function

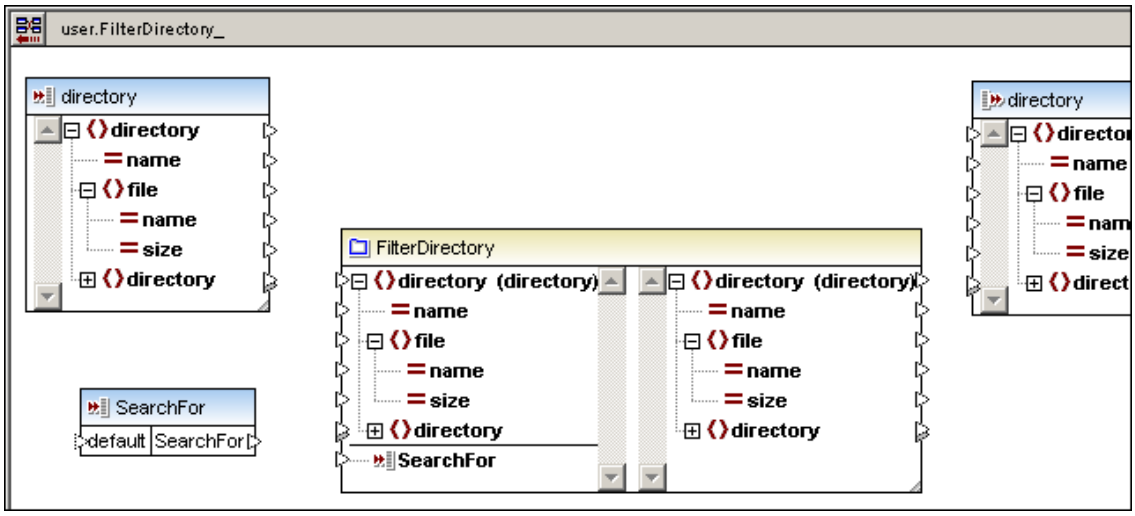
At this point, all the necessary input and output components have been defined for the user-defined function. What we need to do now is insert the "unfinished" function into the current user-defined function window. (You could do this at almost any point however.)

1. Find the FilterDirectory function in the **user** section of the **Libraries** window.
2. Click FilterDirectory then drag and drop it into the FilterDirectory window you have just been working in.



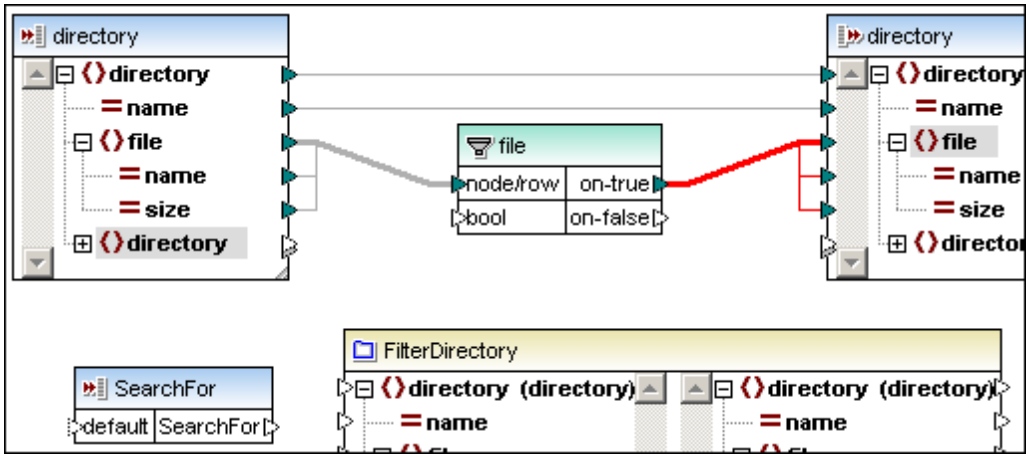
Java Selected

The user-defined function now appears in the user-defined function window as a recursive component.

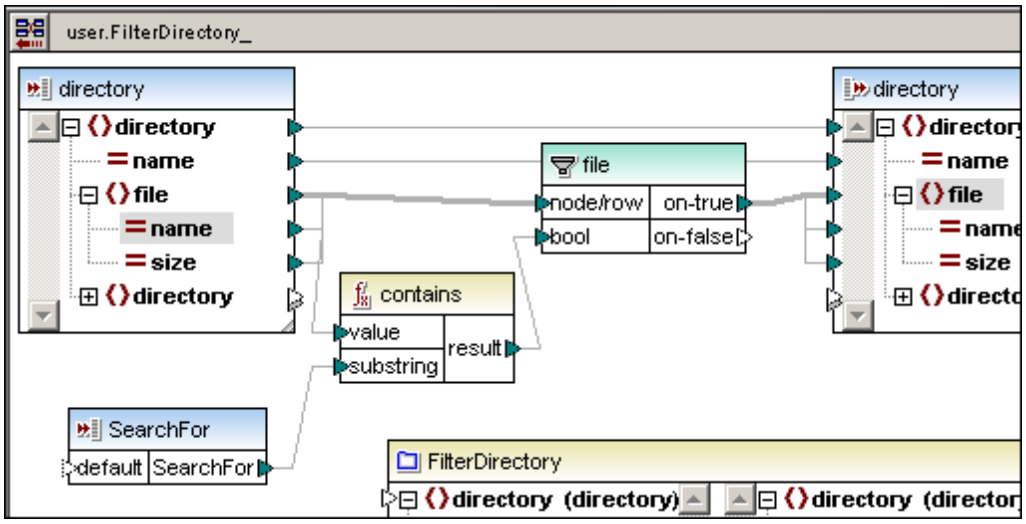


3. Connect the **directory, name** and **file** items of the input component to the same items in the output component.
4. Right click the connector between the **file** items and select "Insert Filter" to insert a filter component.
5. Right click the on-true connector and select **Copy-All** from the context menu.
   The connectors change appearance to Copy-All connectors.

 © 2015 Altova GmbH



6. Insert a Contains function from the **Core | String functions** library.
7. Connect **name** to **value** and the **SearchFor** parameter to **substring**, then result to the bool item of the filter.



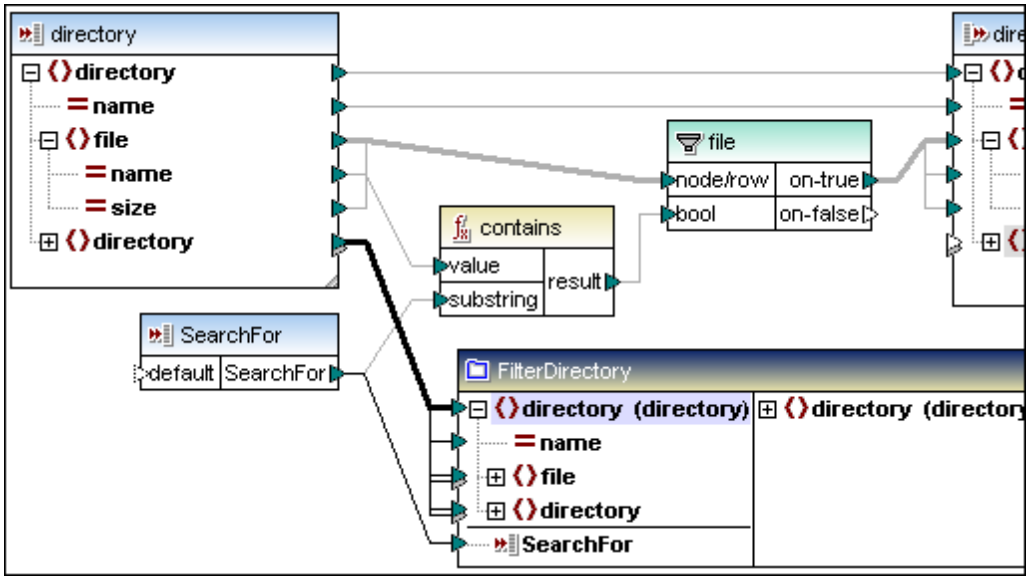
8. Connect the SearchFor item of the input component to the SearchFor item of the user-defined function.

Defining the recursion

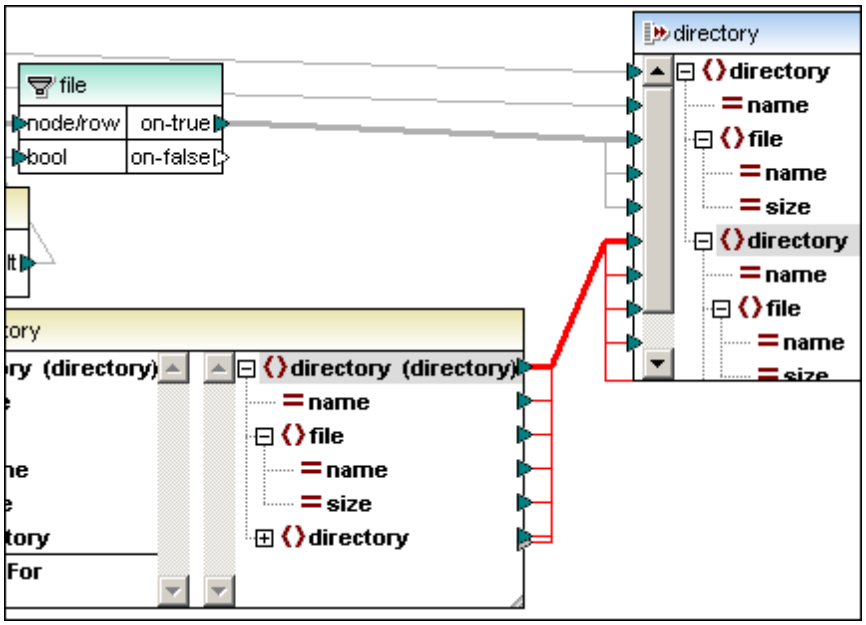
At this point, the mapping of a single directory recursion level is complete. Now we just need to define how to process a subdirectory.

Making sure that the Toggle Autoconnect icon is active in the icon bar:

1. Connect the lower **directory** item of the input component to the top **directory** item of the recursive user-defined function.



2. Connect the top output directory item of the user-defined function to the lower directory item of the output component.
3. Right click the top connector, select Copy-All from the context menu and click OK when prompted if you want to create Copy-All connection.



This completes the definition of the user-defined function in this window.

Click the Return to main mapping window icon, to continue defining the mapping there.

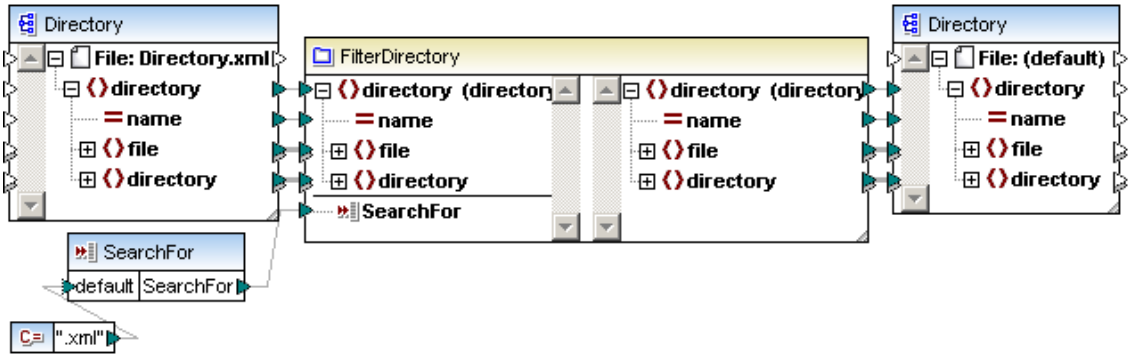
Main Mapping window

1. Drag the FilterDirectory function from the **user** section of the Libraries window, into the **main** mapping area.
2. Use **Insert | XML Schema file** to insert Directory.xsd and select Directory.xml as the instance file.
3. Use the same method to insert Directory.xsd and select Skip, to create the output

 © 2015 Altova GmbH

component.

4. Insert a constant component, then a Input component e.g. SearchFor.
5. Create the connections as shown in the screenshot below.
6. When connecting the top-level connectors, directory to directory, on both sides of the user-defined component, right click the connector and select **Copy-All** from the context menu.



7. Click the Output tab to see the result of the mapping.

Notes:
Double clicking the lowest "directory" item in the Directory component, opens a new level of recursion, i.e. you will see a new **directory | file | directory** sublevel. Using the Copy-all connector automatically uses all existing levels of recursion in the XML instance, you do not need expand the recursion levels manually.

7.3 Adding custom XSLT functions

MapForce allows you to extend the installed XSLT function libraries with your own custom functions. This option is made available when you select XSLT as transformation language.

XSLT files appear as libraries, and display all **named templates** as functions below the library name.

- Functions must be declared as Named Templates conforming to the XSLT specification in the XSLT file.
- If the imported XSLT file imports or includes other XSLT files, then these XSLT files and functions will be imported as well.
- Each named template appears as a function below each library name.
- The amount of mappable input icons depends on the number of parameters used in the template call; optional parameters are also supported.
- Updates to imported XSLT files occur at program start or whenever the files change.
- Namespaces are supported

Note: When writing named templates, make sure that the XPath statements used in the template are bound to the correct namespace(s). To see the namespace bindings of the mapping, preview the generated XSLT code.

See also:

XSLT 1.0 engine implementation
XSLT 2.0 engine implementation

7.3.1 Adding custom XSLT 1.0 functions

The files needed for the simple example shown below, are available in the **...\MapForceExamples** directory.

- Name-splitter.xslt
- Name-splitter.xml (the XML instance file for Customers.xsd)
- Customers.xsd
- CompletePO.xsd

For an additional example of using named templates to sum nodes, see Aggregate functions.

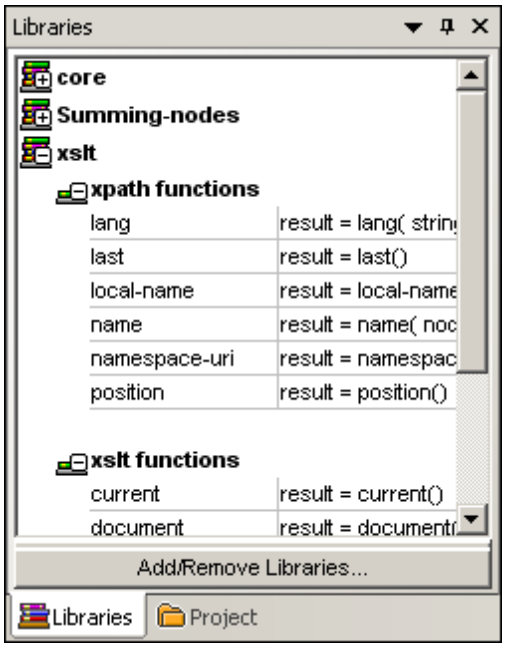
To add a custom XSLT function:

1. Create an XSLT file that achieves the transformation/result you want.
   The example below, **Name-splitter.xslt**, shows a named template called "**tokenize**" with a single parameter "string". What the template does, is work through an input string and separate capitalized characters with a space for each occurrence.

© 2015 Altova GmbH

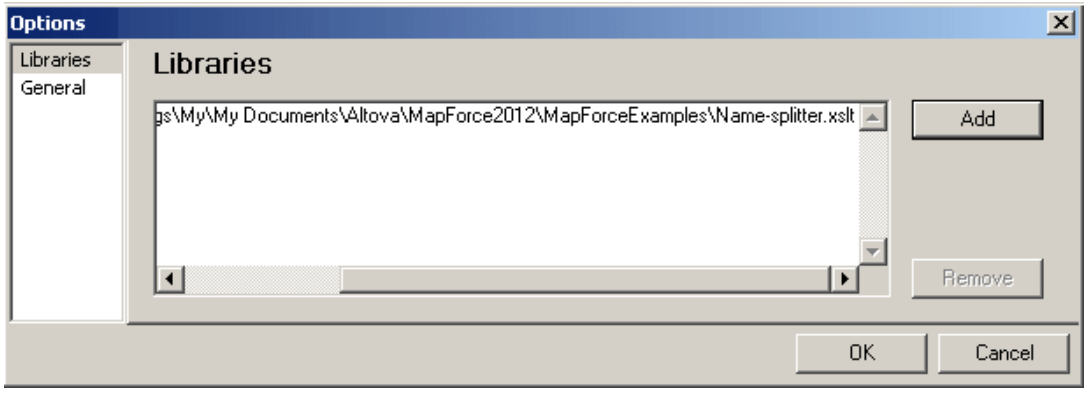
```
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
4
5    <xsl:template match="*">
6      <xsl:for-each select=".">
7      <xsl:call-template name="tokenize">
8      <xsl:with-param name="string" select="."/>
9      </xsl:call-template>
10     </xsl:for-each>
11   </xsl:template>
12
13   <xsl:template name="tokenize">
14     <xsl:param name="string" select="."/>
15     <xsl:variable name="caps" select="translate($string, '-abcdefghijklmnopqrstuv
16     <xsl:variable name="capscount" select="string-length($caps)"/>
17     <xsl:variable name="token">
```

2. Click the **Add/Remove Libraries** button, and then click the Add button in the following dialog box.

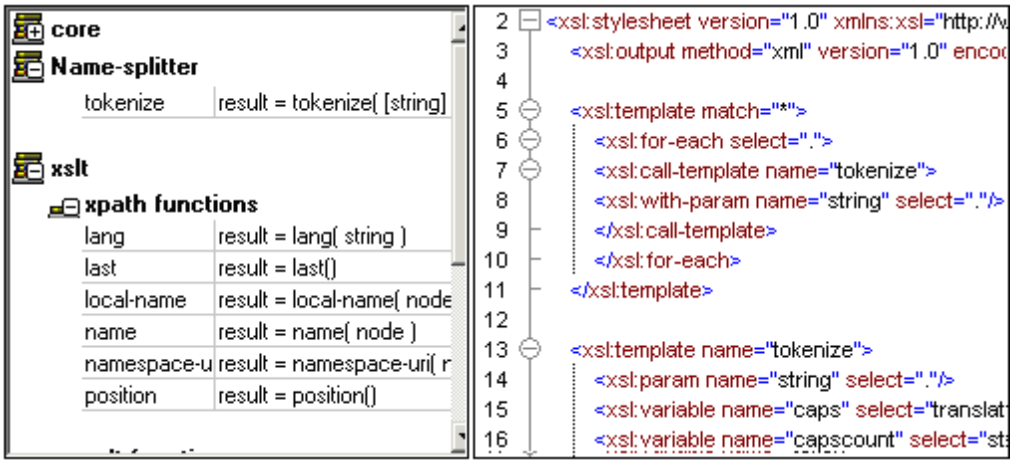


XSLT Selected

3. Select the XSL, or XSLT file, that contains the named template you want to act as a function, in this case **Name-splitter.xslt**. The XSLT file appears in the Libraries tab.



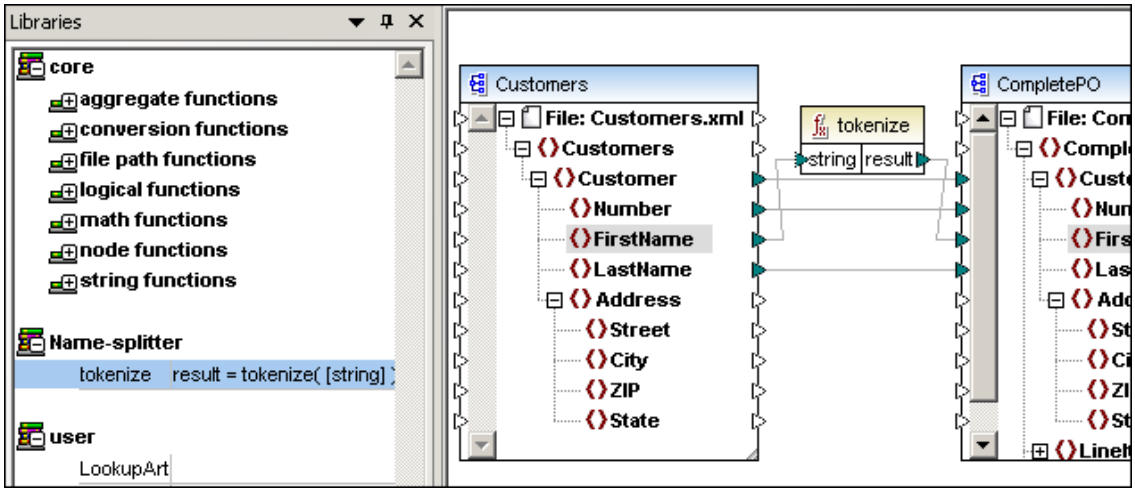
4. Click **OK** to insert the new function.



XSLT Selected

The XSLT file name appears in the library window, along with the function(s) defined as named templates, below it. In this example **Name-splitter** with the **tokenize** function.

5. Drag the function into the Mapping window, to use it in you current mapping, and map the necessary items, as show in the screenshot below.



XSLT Selected

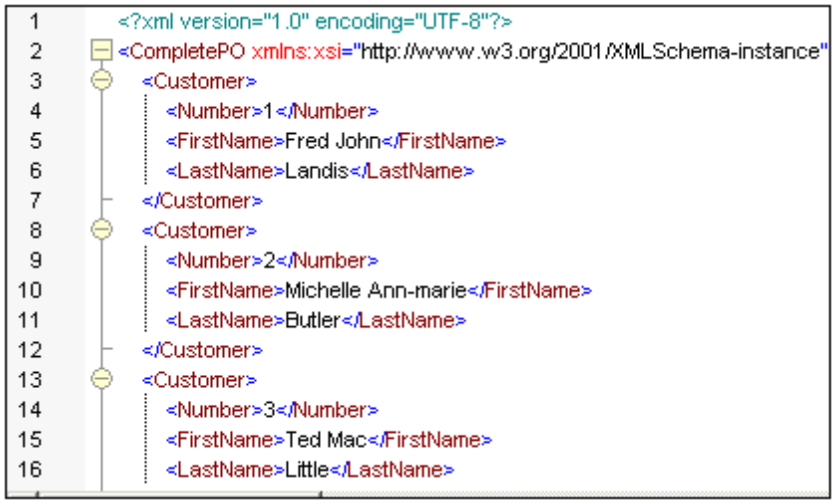
6. Click the XSLT tab to see the generated XSLT code.

 © 2015 Altova GmbH

Please note:

As soon as a named template is used in a mapping, the XSLT file containing the named template is **included** in the generated XSLT code (**xsl:include href...**), and is **called** using the command **xsl:call-template**.

7. Click the Output tab to see the result of the mapping.



To delete custom XSLT functions:

1. Click the **Add/Remove Libraries** button.
2. Click to the specific XSLT **library name** in the Libraries tab
3. Click the Delete button, then click OK to confirm.

7.3.2 Adding custom XSLT 2.0 functions

MapForce also allows you to import XSLT 2.0 functions that occur in an XSLT 2.0 document in the form:

<xsl:function name="MyFunction">

For an additional example of using named templates to sum nodes, see Aggregate functions.

Datatypes in XPath 2.0
If your XML document references an XML Schema and is valid according to it, you must explicitly construct or cast datatypes that are not implicitly converted to the required datatype by an operation.

In the XPath 2.0 Data Model used by the Altova XSLT 2.0 Engine, all **atomized** node values from the XML document are assigned the `xs:untypedAtomic` datatype. The `xs:untypedAtomic` type works well with implicit type conversions.

For example,

- the expression `xs:untypedAtomic("1") + 1` results in a value of 2 because the `xdt:untypedAtomic` value is **implicitly** promoted to `xs:double` by the addition operator.
- Arithmetic operators implicitly promote operands to `xs:double`.
- Value comparison operators promote operands to `xs:string` before comparing.

See also:

XSLT 2.0 engine implementation

7.3.3 Aggregate functions - summing nodes in XSLT1 and 2

This section describes the method you can use to process multiple nodes of an XML instance document and have the result mapped as a single value to a target item. The files used in this example are available in the ...\MapForceExamples\Tutorial\ folder and consists of:

Summing-nodes.mfd	mapping file
input.xml	input XML file
input.xsd and output.xsd	source and target schema files
Summing-nodes.xslt	xslt file containing a named template to sum the individual nodes

There are two separate methods of creating and using aggregate functions:

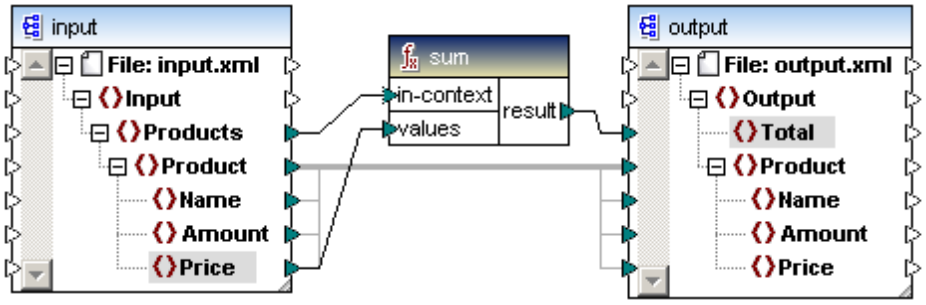
- Using the aggregate functions available in the **core library** of the Library pane
- Using a **Named Template**.

Aggregate functions - library

Depending on the XSLT library you select, XSLT 1 or XSLT 2, different aggregate functions are available in the core library. XSLT 1 supports count and sum, while XSLT 2 supports avg, count, max, min, string-join and sum.

© 2015 Altova GmbH

Drag the aggregate function that you use from the library into the mapping area and connect the source and target components as shown in the screenshot below.



For more information on this type of aggregate function, please also see Aggregate functions.

Aggregate function - Named template

The screenshot below shows the **XML input** file. The aim of the example is to sum the Price fields of any number of products, in this case products A and B.

```
<?xml version="1.0" encoding="UTF-8"?>
<Input xmlns:xsi="http://www.w3.org/2001/XMLSchema
    <Products>
        <Product>
            <Name>ProductA</Name>
            <Amount>10</Amount>
            <Price>5</Price>
        </Product>
        <Product>
            <Name>ProductB</Name>
            <Amount>5</Amount>
            <Price>20</Price>
        </Product>
    </Products>
</Input>
```

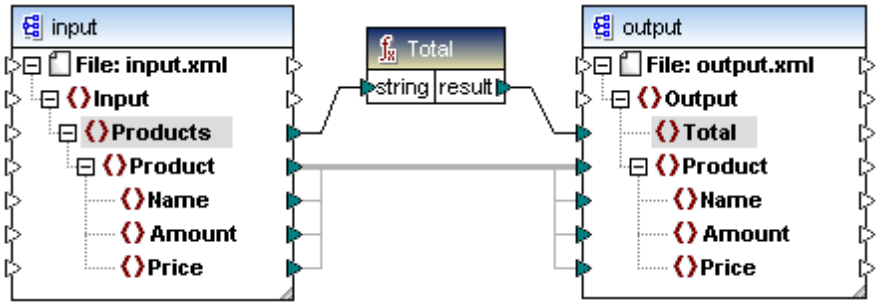
The screenshot below shows the XSLT stylesheet which uses the named template "**Total**" and a single parameter "string". What the template does, is work through the XML input file and sum all the values obtained by the XPath expression **/Product/Price**, in the document.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/19
    <xsl:output method="xml" version="1.0" encoding="UTF-8" i

    <xsl:template match="*">
        <xsl:for-each select=".">
        <xsl:call-template name="Total">
            <xsl:with-param name="string" select="."/>
        </xsl:call-template>
        </xsl:for-each>
    </xsl:template>


    <xsl:template name="Total">
    <xsl:param name="string"/>
        <xsl:value-of select="sum($string/Product/Price)"/>
    </xsl:template>
</xsl:stylesheet>
```

1. Click the **Add/Remove Libraries** button, and select the Libraries tab of the Options dialog box.
2. Click the Add button and select the **Summing-nodes.xslt** file from the **<Documents>\Altova\MapForce2016\MapForceExamples** folder.
3. Drag in the Total function from the newly created Summing-nodes library and create the mappings as shown below.



4. Click the Output tab to preview the mapping result.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Output xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNa
3    <Total>25</Total>
4    <Product>
5      <Name>ProductA</Name>
6      <Amount>10</Amount>
7      <Price>5</Price>
8    </Product>
9    <Product>
10     <Name>ProductB</Name>
11     <Amount>5</Amount>
12     <Price>20</Price>
13   </Product>
14 </Output>
15
```

The two Price fields of both products have been added and placed into the Total field.

To sum the nodes in XSLT 2.0:

- Change the stylesheet declaration in the template to ... version="2.0".

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet version="2.0" xmlns:xs
3      <xsl:output method="xml" version="
```


© 2015 Altova GmbH

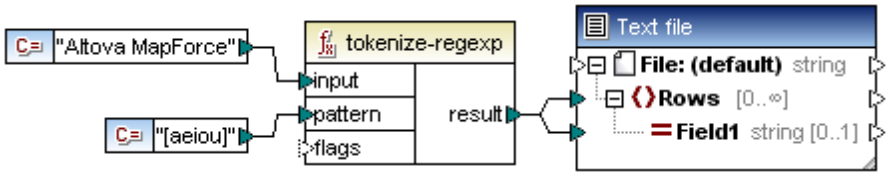
7.4 Regular Expressions

MapForce can use regular expressions in the **pattern** parameter of the **tokenize-regexp** function, to find specific strings.

The regular expression syntax and semantics for XSLT and XQuery are identical to those defined in http://www.w3.org/TR/xmlschema-2/. Please note that there are slight differences in regular expression syntax between the various programming languages.

Terminology

input	the string that the regex works on
pattern	the regular expression
flags	optional parameter to define how the regular expression is to be interpreted
result	the result of the function



Tokenize-regexp returns a sequence of strings. The connection to the Rows item creates one row per item in the sequence.

regex syntax

Literals e.g. a single character:
e.g. The letter "a" is the most basic regex. It matches the first occurrence of the character "a" in the string.

Character classes []

This is a set of characters enclosed in square brackets.

One, and only one, of the characters in the square brackets are matched.

pattern **[aeiou]**
Matches a lowercase vowel.

pattern **[mj]ust**
Matches must or just

Please note that "pattern" is case sensitive, a lower case **a** does not match the uppercase **A.**

Character ranges **[a-z]**
Creates a range between the two characters. Only one of the characters will be matched at one time.

pattern **[a-z]**
Matches any lowercase characters between a and z.

negated classes **[^]**
using the caret as the first character after the opening bracket, negates the character class.

pattern **[^a-z]**
Matches any character not in the character class, including newlines.

Meta characters "."

Dot meta character
matches **any single** character (except for newline)

pattern **.**
Matches any single character.

Quantifiers ? + * {}

Quantifiers define how often a regex component must repeat within the input string, for a match to occur.

?	
zero or one	preceding string/chunk is optional
+	
one or more	preceding string/chunks may match one or more times
*****	
zero or more	preceding string/chunks may match zero or more times
{}	
min / max repetitions	no. of repetitions a string/chunks has to match e.g. mo{1,3} matches mo, moo, mooo.

()
subpatterns
parentheses are used to group parts of a regex together.

|
Alternation/or allows the testing of subexpressions form left to right.
(horse|make) sense - will match "horse sense" or "make sense"

flags

These are optional parameters that define how the regular expression is to be interpreted. Individual letters are used to set the options, i.e. the character is present. Letters may be in any order and can be repeated.

s

 © 2015 Altova GmbH

If present, the matching process will operate in the "dot-all" mode.

The meta character "**.**" matches any character whatsoever. If the input string contains "hello" and "world" on two *different* lines, the regular expression "hello*world" will only match if the **s** flag/ character is set.

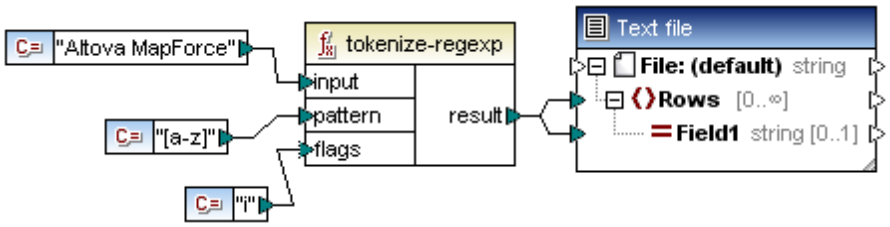
m
If present, the matching process operates in multi-line mode.

In multi-line mode the caret **^** matches the start of **any** line, i.e. the start of the entire string **and** the first character after a newline character.

The dollar character **$** matches the end of **any** line, i.e. the end of the entire string and the character immediately before a newline character.

Newline is the character #x0A.

i
If present, the matching process operates in case-insensitve mode.
The regular expression [a-z] plus the **i** flag would then match all letters a-z and A-Z.



x
If present, whitespace characters are removed from the regular expression prior to the matching process. Whitespace chars. are #x09, #x0A, #x0D and #x20.

Exception:
Whitespace characters within character class expressions are not removed e.g. [#x20].

Please note:
When generating code, the advanced features of the regex syntax might differ slightly between the various languages, please see the specific regex documentation for your language.

7.5 Function Library Reference

This reference chapter describes the MapForce built-in functions available in the Libraries pane, organized by library.

The availability of function libraries in the Libraries pane depends on the transformation language you have selected (see Selecting a transformation language).

XPath 2.0 restrictions: Several XPath 2.0 functions dealing with sequences are currently not available.

7.5.1 core | aggregate functions

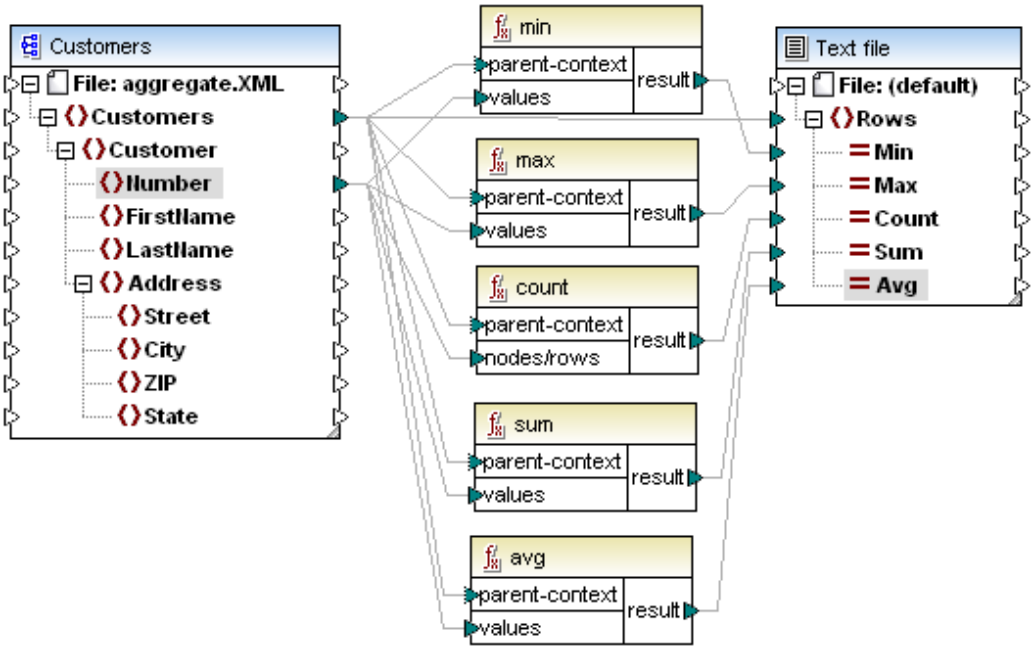
Aggregate functions perform operations on a set, or sequence, of input values. The input data for min, max, sum and avg is converted to the **decimal** data type for processing.

- The input values must be connected to the **values** parameter of the function.
- A context node (item) can be connected to the **parent-context** parameter to override the default context from which the input sequence is taken. The parent-context parameter is optional.
- The **result** of the function is connected to the specific target item.

The mapping shown below is available as **Aggregates.mfd** in the ...\Tutorial folder and shows how these functions are used.

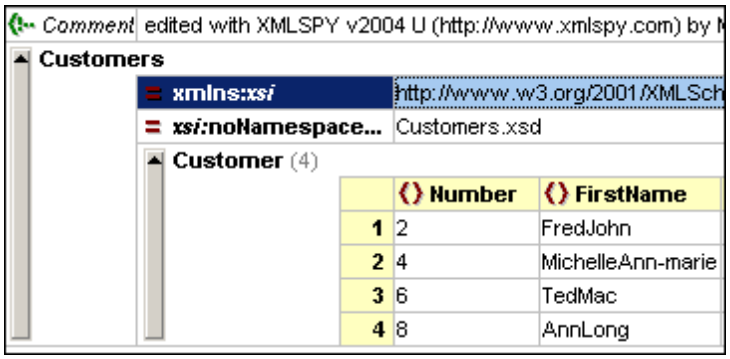
Aggregate functions have two input items.

- **values** (nodes/rows) is connected to the source item that provides the data, in this case Number.
- **parent-context** is connected to the item you want to iterate over, i.e. the context, in this case over all Customers. The parameter is, however, optional.

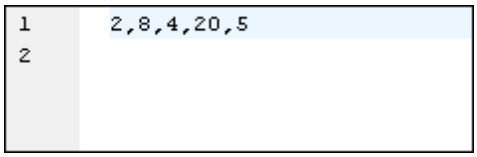


 © 2015 Altova GmbH

The input instance in this case is an XML file containing the following data:



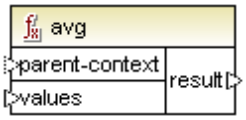
- The source data supplied to the values item is the number sequence 2,4,6,8.
- The output component in this case is a simple text file.
  Clicking the Output tab for the above mapping delivers the following result:



min=2, max=8, count=4, sum=20 and avg=5.

7.5.1.1 avg

Returns the average value of all values within the input sequence. The average of an empty set is an empty set. Not available in XSLT1.

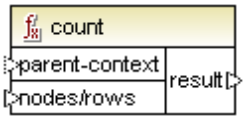


Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the context.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric.

For an example of usage, see the mapping **GroupTemperaturesByYear.mfd** in the **<Documents>\Altova\MapForce2016\MapForceExamples** directory.

7.5.1.2 count

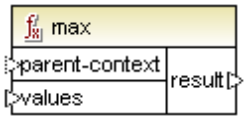
Returns the number of individual items making up the input sequence. The count of an empty set is zero. Limited functionality in XSLT1.



Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the context.
nodes/rows	This argument must be connected to the source item to be counted.

7.5.1.3 max

Returns the maximum value of all numeric values in the input sequence. Note that this function returns an empty set if the **strings** argument is an empty set. Not available in XSLT1.

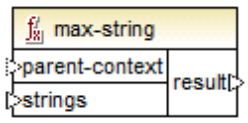


Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the context.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric. To get the maximum from a sequence of strings, use the **`max-string`** function.

For an example of usage, see the mapping **GroupTemperaturesByYear.mfd** in the **<Documents>\Altova\MapForce2016\MapForceExamples** directory.

7.5.1.4 max-string

Returns the maximum value of all string values in the input sequence. For example, `max-string("a", "b", "c")` returns "c". This function is not available in XSLT1.



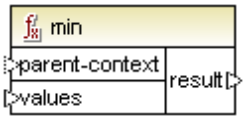
Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the context.

Argument	Description
strings	This argument must be connected to a source item which supplies the actual data. The supplied argument value must be a sequence (zero or many) of `xs:string`.

Note that the function returns an empty set if the **strings** argument is an empty set.

7.5.1.5 min

Returns the minumum value of all numeric values in the input sequence. The minimum of an empty set is an empty set. Not available in XSLT1.

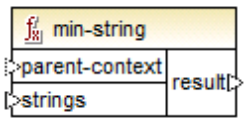


Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the context.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric. To get the minimum from a sequence of strings, use the **`min-string`** function.

For an example of usage, see the mapping **GroupTemperaturesByYear.mfd** in the **<Documents>\Altova\MapForce2016\MapForceExamples** directory.

7.5.1.6 min-string

Returns the minimum value of all string values in the input sequence. For example, `min-string("a", "b", "c")` returns "a". This function is not available in XSLT1.

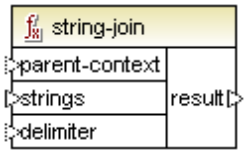


Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the context.
strings	This argument must be connected to a source item which supplies the actual data. The supplied argument value must be a sequence (zero or many) of `xs:string`.

Note that the function returns an empty set if the **strings** argument is an empty set.

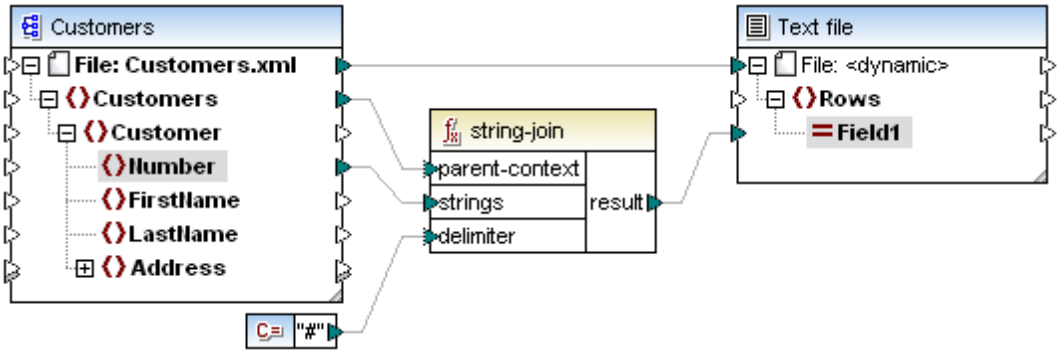
7.5.1.7 string-join

Concatenates all the values of the input sequence into one string delimited by whatever string you choose to use as the delimiter. The string-join of an empty set is the empty string. Not available in XSLT1.



The example below contains four separate customer numbers 2 4 6 and 8. The constant character supplies a hash character "#" as the delimiter.

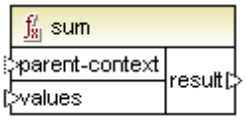
Result = 2#4#6#8



If you do not supply a delimiter, then the default is an empty string, i.e. no delimiter of any sort. Result = 2468.

7.5.1.8 sum

Returns the arithmetic sum of all values in the input sequence. The sum of an empty set is zero. Not available in XSLT1.



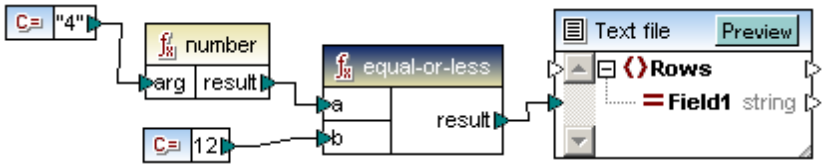
Argument	Description
parent-context	Optional argument. Supplies the parent context. See also Overriding the context.
values	This argument must be connected to a source item which supplies the actual data. Note that the supplied argument value must be numeric.

 © 2015 Altova GmbH

7.5.2 core | conversion functions

To support explicit data type conversion, several type conversion functions are available in the **conversion** library. Note that, in most cases, MapForce creates necessary conversions automatically and these functions need to be used only in special cases.

If the input nodes are of differing types, e. g. integer and string, you can use the conversion functions to force a string or numeric comparison.



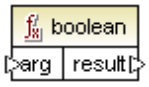
In the example above the first constant is of type string and contains the string "4".
The second constant contains the numeric constant 12. To be able to compare the two values explicitly the types must agree.

Adding a **number** function to the first constant converts the string constant to the numeric value of 4. The result of the comparisons is then "true".

Note that if the number function were not be used, i.e 4 would be connected directly to the **a** parameter, a string compare would occur, with the result being false.

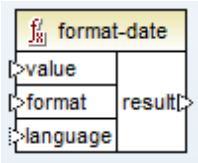
7.5.2.1 boolean

Converts an input numeric value into a boolean (as well as a string to numeric - true to 1). E.g. 0 to "false", or 1 to "true", for further use with logical functions (equal, greater etc.) filters, or if-else functions.



7.5.2.2 format-date

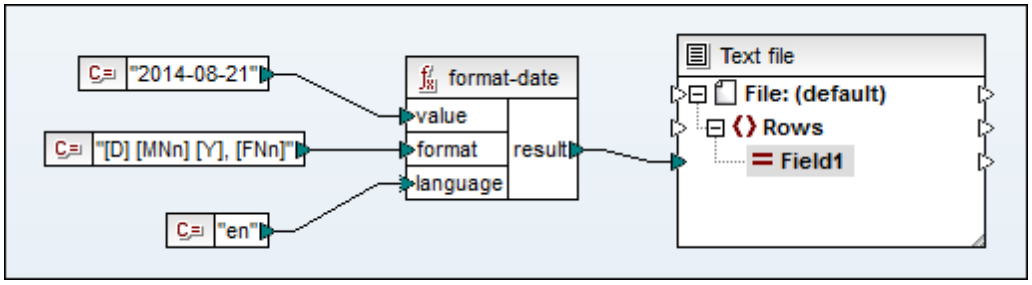
Converts an `xs:date` input value into a string and formats it according to specified options.



Argument	Description
value	The date to be formatted.

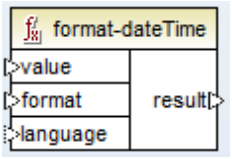
Argument	Description
format	A format string identifying the way in which the date is to be formatted. This argument is used in the same way as the `format` argument in the **`format-dateTime`** function.
language	Optional argument. When supplied, the name of the month and the day of the week are returned in a specific language. Valid values: **en (default)** English **es** Spanish **de** German **ja** Japanese

In the following example, the output result is: "21 August 2014, Thursday". To translate this value to Spanish, set the value of the language argument to `es`.



7.5.2.3 *format-dateTime*

Converts a dateTime into a string.



Argument	Description
value	The `xs:dateTime` value to be formatted.
format	A format string identifying the way in which **value** is to be formatted.
language	Optional argument. When supplied, the name of the month and the day of the week are returned in a specific language. Valid values: **en (default)** English **es** Spanish

 © 2015 Altova GmbH

Argument	Description	
	de	German
	ja	Japanese

Note:
If the function's output (i.e. result) is connected to a node of type other than string, the formatting may be lost as the value is cast to the target type. This automatic cast can be disabled by unchecking the **Cast target values to target types** check box in the Component Settings of the target component.

The format argument consists of a string containing so-called variable markers enclosed in square brackets. Characters outside the square brackets are literal characters to be copied into the result. If square brackets are needed as literal characters in the result, then they should be doubled.

Each variable marker consists of a component specifier identifying which component of the date or time is to be displayed, an optional formatting modifier, another optional presentation modifier and an optional width modifier, preceded by a comma if it is present.

```
format := (literal | argument)*
argument := [component(format)?(presentation)?(width)?]
width := , min-width ("-" max-width)?
```

The components are:

Specifier	Description	Default Presentation
Y	year (absolute value)	four digits (2010)
M	month of the year	1-12
D	day of month	1-31
d	day of year	1-366
F	day of week	name of the day (language dependent)
W	week of the year	1-53
w	week of month	1-5
H	hour (24 hours)	0-23
h	hour (12 hour)	1-12
P	A.M. or P.M.	alphabetic (language dependent)
m	minutes in hour	00-59
s	seconds in minute	00-59
f	fractional seconds	numeric, one decimal place

Specifier	Description	Default Presentation
Z	timezone as a time offset from UTC	+08:00
z	timezone as a time offset using GMT	GMT+n

The formatting modifier:

Character	Description	Example
1	decimal numeric format with no leading zeros: 1, 2, 3, ...	1, 2, 3
01	decimal format, two digits: 01, 02, 03, ...	01, 02, 03
N	name of component, upper case	MONDAY, TUESDAY [1)]
n	name of component, lower case	monday, tuesday [1)]
Nn	name of component, title case	Monday, Tuesday [1)]

Note: N, n, and Nn modifiers only support the following components: M, d, D.

The width modifier, if present, is introduced by a comma. It takes the form:

```
, min-width ("-" max-width)?
```

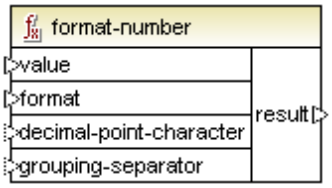
Supported examples

DateTime	format String	Result
2003-11-03T00:00:00	`[D]/[M]/[Y]`	3/11/2003
2003-11-03T00:00:00	`[Y]-[M,2]-[D,2]`	2003-11-03
2003-11-03T00:00:00	`[Y]-[M,2]-[D,2]`	2003-11-03
2003-11-03T00:00:00	`[Y]-[M,2]-[D,2] [H,2]:[m]:[s]`	2003-11-03 00:00:00
2010-06-02T08:02	`[Y] [MNn] [D01] [F,3-3] [d] [H]:[m]:[s].[f]`	2010 June 02 Wed 153 8:02:12.054
2010-06-02T08:02	`[Y] [MNn] [D01] [F,3-3] [d] [H]:[m]:[s].[f] [z]`	2010 June 02 Wed 153 8:02:12.054 GMT+02:00
2010-06-02T08:02	`[Y] [MNn] [D1] [F] [H]:[m]:[s].[f] [Z]`	2010 June 2 Wednesday 8:02:12.054 +02:00
2010-06-02T08:02	`[Y] [MNn] [D] [F,3-3] [H01]:[m]:[s]`	2010 June 2 Wed 08:02:12

© 2015 Altova GmbH

7.5.2.4 format-number

Converts a number into a string. The function is available for XSLT 1.0, XSLT 2.0, Java, C#, C++ and Built-in execution engine.



Argument	Description
`value`	Mandatory argument. Supplies the number to be formatted.
`format`	Mandatory argument. Supplies a format string that identifies the way in which the number is to be formatted. This argument is used in the same way as the `format` argument in the **`format-dateTime`** function.
`decimal-point-format`	Optional argument. Supplies the character to be used as the decimal point character. The default value is the full stop (`.`) character.
`grouping-separator`	Optional argument. Supplies the character used to separate groups of numbers. The default value is the comma (`,`) character.

Note: If the function's output (i.e. result) is connected to a node of type other than string, the formatting may be lost as the value is cast to the target type. This automatic cast can be disabled by unchecking the **Cast target values to target types** check box in the component settings of the target component.

Format:

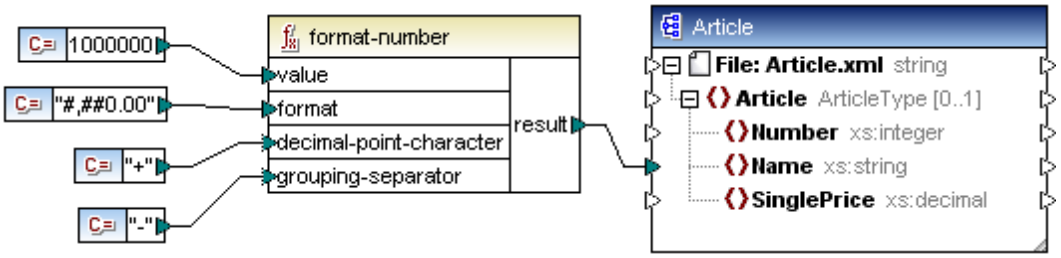
```
format := subformat (;subformat)?
  subformat := (prefix)? integer (.fraction)? (suffix)?
  prefix := any characters except special characters
  suffix := any characters except special characters
  integer := (#)* (0)* ( allowing ',' to appear)
  fraction := (0)* (#)* (allowing ',' to appear)
```

The first *subformat* is used for formatting positive numbers, and the second subformat for negative numbers. If only one *subformat* is specified, then the same subformat will be used for negative numbers, but with a minus sign added before the *prefix*.

Special Character	default	Description
zero-digit	0	A digit will always appear at this point in the result

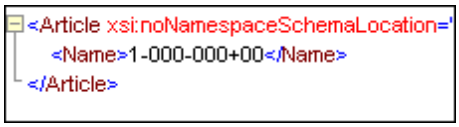
Special Character	default	Description
digit	#	A digit will appear at this point in the result string unless it is a redundant leading or trailing zero
decimal-point	.	Separates the integer and the fraction part of the number.
grouping-seperator	,	Seperates groups of digits.
percent-sign	%	Multiplies the number by 100 and shows it as a percentage.
per-mille	‰	Multiplies the number by 1000 and shows it as per-mille.

The characters used for decimal-point-character and grouping-separator are always "." and "," respectively. They can, however, be changed in the formatted output, by mapping constants to these nodes.



The result of the format number function shown above.

- The decimal-point character was changed to a "+".
- The grouping separator was changed to a "-"



Rounding

The rounding method used for this function is "half up", e.g. the value gets rounded up if the fraction is greater than or equal to 0.5. The value gets rounded down if the fraction is less than 0.5. This method of rounding only applies to generated code and the built-in execution engine.

In XSLT 1.0, the rounding mode is undefined. In XSLT 2.0, the rounding mode is "round-half-to-even".

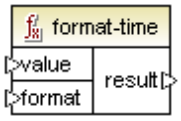
Number	Format String	Result
1234.5	#,##0.00	1,234.50
123.456	#,##0.00	123.46

 © 2015 Altova GmbH

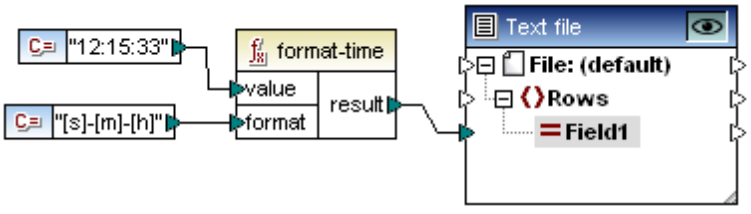
Number	Format String	Result
1000000	#,##0.00	1,000,000.00
-59	#,##0.00	-59.00
1234	###0.0###	1234.0
1234.5	###0.0###	1234.5
.00025	###0.0###	0.0003
.00035	###0.0###	0.0004
0.25	#00%	25%
0.736	#00%	74%
1	#00%	100%
-42	#00%	-4200%
-3.12	#.00;(#.00)	(3.12)
-3.12	#.00;#.00CR	3.12CR

7.5.2.5 *format-time*

Converts an xs:time input value into a string. The `format` argument is used in the same way as the `format` argument in the **`format-dateTime`** function.



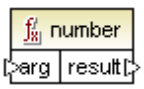
E.g



Result: 33-15-12

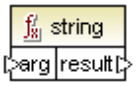
7.5.2.6 *number*

Converts an input string into a number. Also converts a boolean input to a number.



7.5.2.7 string

Converts an input value into a string. The function can also be used to retrieve the text content of a node.



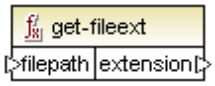
If the input node is a XML complex type, then all descendents are also output as a single string.

7.5.3 core | file path functions

The **file path** functions allow you to directly access and manipulate file path data, i.e. folders, file names, and extensions for further processing in your mappings. They can be used in all languages supported by MapForce.

7.5.3.1 get-fileext

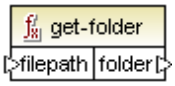
Returns the extension of the file path including the dot "." character.



E.g. 'c:\data\Sample.mfd' returns '.mfd'

7.5.3.2 get-folder

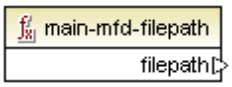
Returns the folder name of the file path including the trailing slash, or backslash character.



E.g. 'c:/data/Sample.mfd' returns 'c:/data/'

7.5.3.3 main-mfd-filepath

Returns the full path of the mfd file containing the main mapping. An empty string is returned if the mfd is currently unsaved.



 © 2015 Altova GmbH

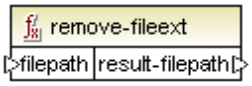
7.5.3.4 mfd-filepath

If the function is called in the main mapping, it returns the same as main-mfd-filepath function, i.e. the full path of the mfd file containing the main mapping. An empty string is returned if the mfd is currently unsaved.

If called within an **user-defined function** which is **imported** by a mfd-file, it returns the full path of the imported mfd file which contains the **definition** of the user-defined function.

7.5.3.5 remove-fileext

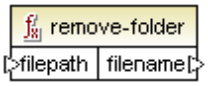
Removes the extension of the file path including the dot-character.



E.g. 'c:/data/Sample.mfd' returns 'c:/data/Sample'.

7.5.3.6 remove-folder

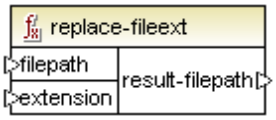
Removes the directory of the file path including the trailing slash, or backslash character.



E.g. 'c:/data/Sample.mfd' returns 'Sample.mfd'.

7.5.3.7 replace-fileext

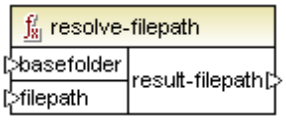
Replaces the extension of the file path supplied by the filepath parameter, with the one supplied by the connection to the extension parameter.



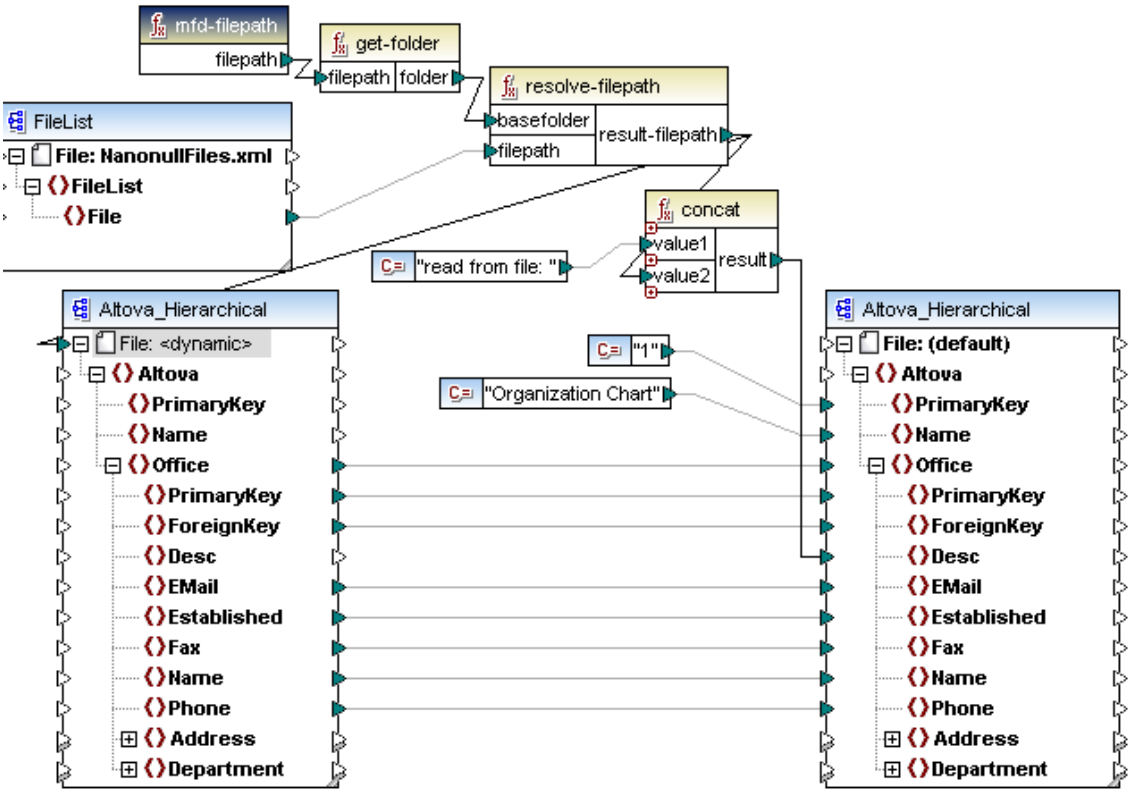
E.g. c:/data/Sample.**mfd'** as the input filepath, and '.**mfp'** as the extension, returns 'c:/data/Sample.mfp'

7.5.3.8 resolve-filepath

Resolves a relative file path to a relative, or absolute, base folder. The function supports '.' (current directory) and '..' (parent directory).



For an example, see the mapping **MergeMultipleFiles_List.mfd** available in the ... \MapForceExamples folder.



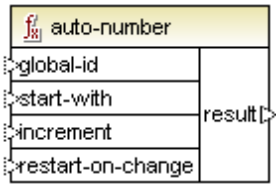
7.5.4 core | generator functions

The **core / generator** functions library includes functions which generate values.

7.5.4.1 auto-number

The **auto-number** function generates integers in target nodes of a component, depending on the various parameters you define. The function result is a value starting at **start_with** and increased by **increment**. Default values are: start-with=1 and increase=1. Both parameters can be negative.

© 2015 Altova GmbH

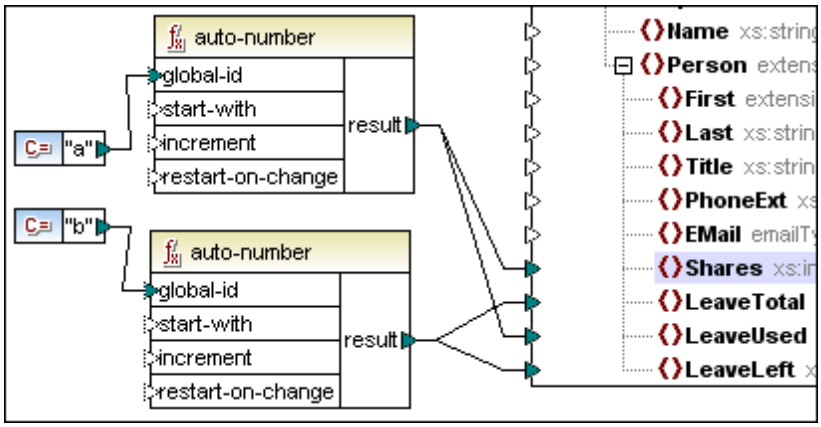


Make sure that the result connector (of the auto-number function) is **directly** connnected to a target node. The exact order in which functions are called by the generated mapping code is undefined. MapForce may choose to cache calculated results for reuse, or evaluate expressions in any order. It is therefore strongly recommended to take care when using the auto-number function.

global-id
This parameter allows you to synchronize the number sequence output of two separate auto-number functions connected to a single target component.

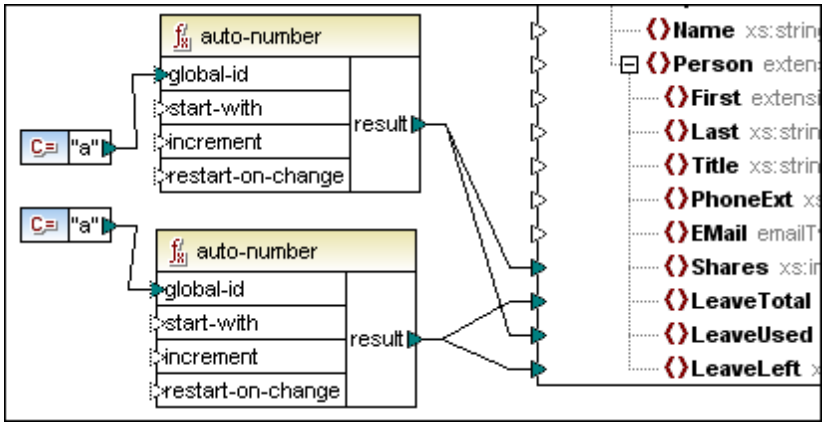
If the two auto-number functions do **not** have the same global-id, then each increments the target items separately. In the example below, each function has a different global-id i.e. a and b.

The output of the mapping is 1,1,2,2. The top function supplies the first 1 and the lower one the second 1.



If both functions have **identical** global-ids, **a** in this case, then each function "knows" about the current auto-number state (or actual value) of the other, and both numbers are then synchronised/ in sequence.

The output of the mapping is therefore 1, 2, 3, 4.The top function supplies the first 1 and the lower one now supplies a 2.

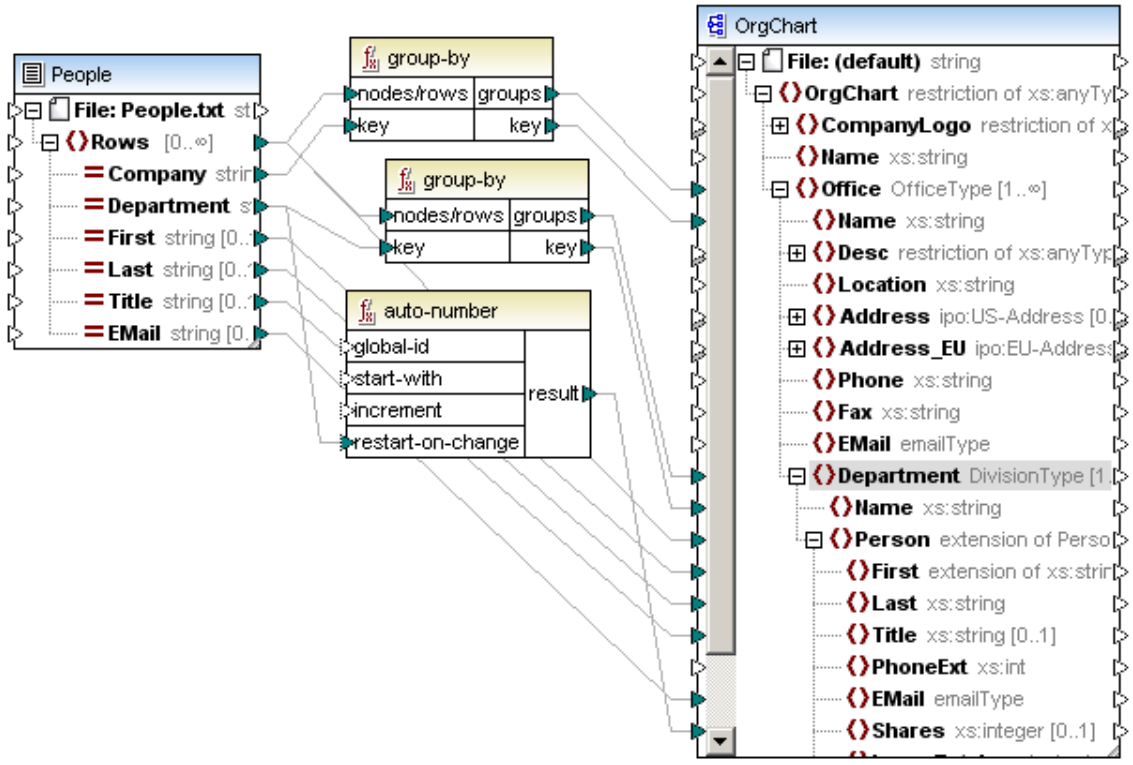


start-with
The inital value used to start the auto numbering sequence. Default is 1.

increment
The increment you want auto-number sequence to increase by. Default is 1.

restart on change
Resets the auto-number counter to "start-with", when the **content** of the connected item changes.

In the example below, start-with and increment are both using the default 1. As soon as the **content** of Department changes, i.e. the department name changes, the counter is reset and starts at 1 for each new department.



 © 2015 Altova GmbH

7.5.5 core | logical functions

Logical functions are (generally) used to compare input data with the result being a boolean "true" or " false". They are generally used to test data before passing on a subset to the target component using a filter.

input parameters = **a** | **b**, or **value1** | **value2**
output parameter = result

The evaluation result of two input nodes depends on the input values as well as the data types used for the comparison.

For example, the 'less than' comparison of the integer values 4 and 12 yields the boolean value "true", since 4 is less than 12. If the two input strings contain '4' and '12', the lexical analysis results in the output value "false", since '4' is alphabetically greater than the first character '1' of the second operand (12).

If all input data types are of the same type, e.g. all input nodes are numerical types, or strings, then the comparison is done for the common type.

If the input nodes are of differing types (for example, integer and string, or string and date), then the data type used for the comparison **is the most general (least restrictive) input data type** of the two input types.

Before comparing two values, all input values are converted to a common datatype. Using the previous example; the datatype "string" is less restrictive than "integer". Comparing integer value 4 with the string '12', converts integer value 4 to the string '4', which is then compared with the string '12'.

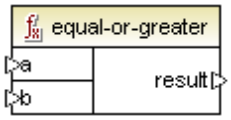
Note: Logical functions cannot be used to test the existence of null values. If you supply a null value as argument to a logical function, it returns a null value. For more information about handling null values, see Nil Values / Nillable.

7.5.5.1 equal

Result is true if a=b, else false.

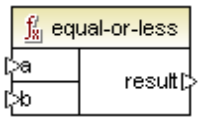
7.5.5.2 equal-or-greater

Result is true if a is equal/greater than b, else false.



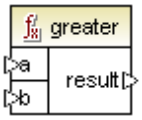
7.5.5.3 equal-or-less

Result is true if a is equal/less than b, else false.



7.5.5.4 greater

Result is true if a is greater than b, else false.

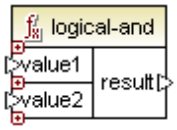


7.5.5.5 less

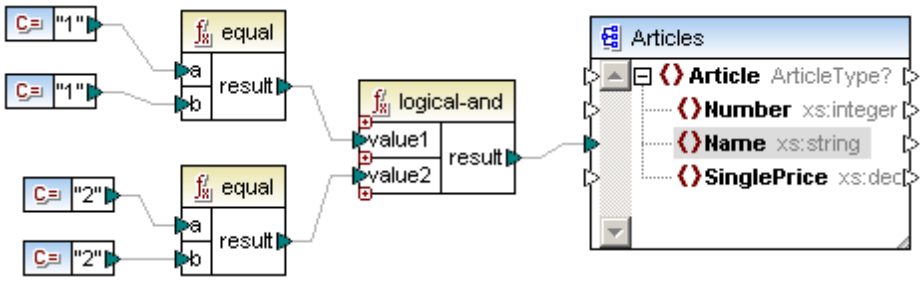
Result is true if a is less than b, else false.

7.5.5.6 logical-and

If **both** value1 and value2 of the logical-and function are **true**, then result is true; if different then false.

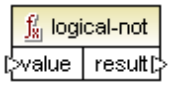


 © 2015 Altova GmbH

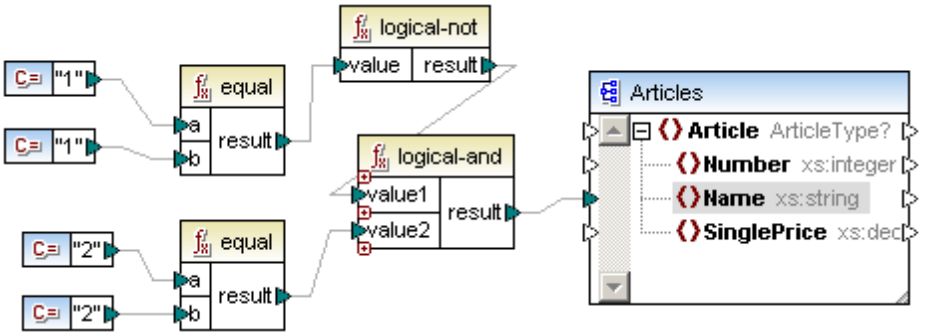


7.5.5.7 *logical-not*

Inverts or flips the logical state/result; if input is true, result of logical-not function is false. If input is false then result is true.

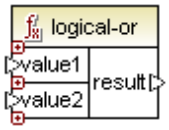


The logical-not function shown below, inverts the result of the equal function. The logical-and function now only returns true if boolean values of value1 and value2 are different, i.e. true-false, or false-true.



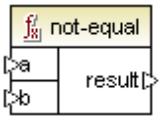
7.5.5.8 *logical-or*

Requires both input values to be boolean. If **either** value1 or value2 of the logical-or function are **true**, then the result is true. If both values are false, then result is false.



7.5.5.9 *not-equal*

Result is true if a is not equal to b.

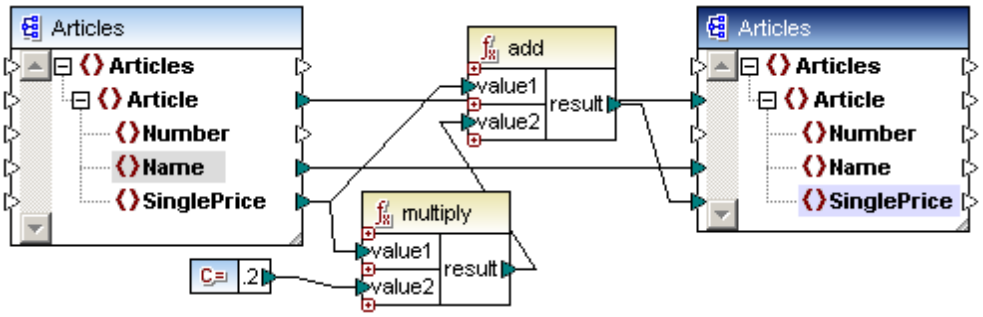


7.5.6 core | math functions

Math functions are used to perform basic mathematical functions on data. Note that they cannot be used to perform computations on durations, or datetimes.

input parameters = **value1** | **value2**
output parameter = **result**

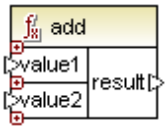
input values are automatically converted to decimal for further processing.



The example shown above, adds 20% sales tax to each of the articles mapped to the target component.

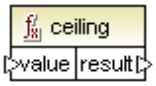
7.5.6.1 add

Result is the decimal value of adding **value1** to **value2**.



7.5.6.2 ceiling

Result is the smallest integer that is greater than or equal to **value**, i.e. the next highest integer value of the decimal input **value**.

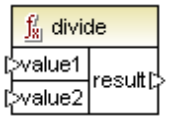


E.g. if the result of a division function is 11.2, then applying the ceiling function to it makes the result 12, i.e. the next highest whole number.

 © 2015 Altova GmbH

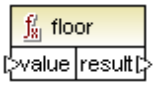
7.5.6.3 divide

Result is the decimal value of dividing **value1** by **value2**. The result precision depends on the target language. Use the round-precision function to define the precision of result.



7.5.6.4 floor

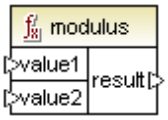
Result is the largest integer that is less than or equal to **value**, i.e. the next lowest integer value of the decimal input **value**.



E.g. if the result of a division function is 11.2, then applying the floor function to it makes the result 11, i.e. the next lowest whole number.

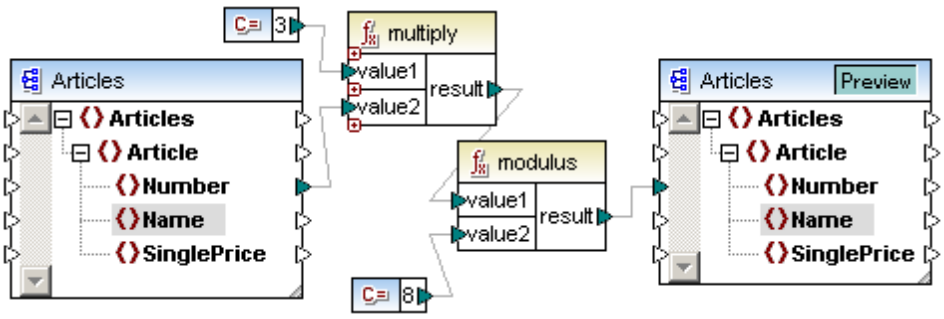
7.5.6.5 modulus

Result is the remainder of dividing **value1** by **value2**.



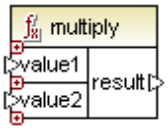
In the mapping below, the numbers have been multiplied by 3 and passed on to value1 of the modulus function. Input values are now 3, 6, 9, and 12.

When applying/using modulus 8 as value2, the remainders are 3, 6, 1, and 4.



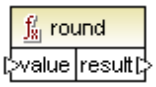
7.5.6.6 multiply

Result is the decimal value of multiplying **value1** by **value2**.



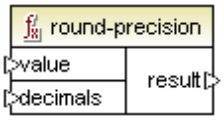
7.5.6.7 round

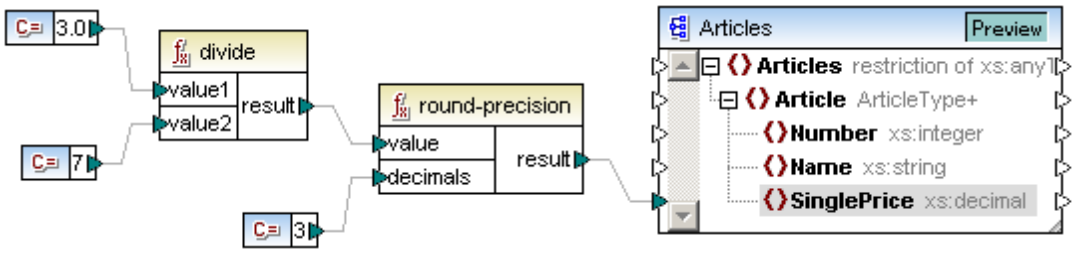
Returns the value rounded to the nearest integer. When the value is exactly in between two integers, the "Round Half Towards Positive Infinity" algorithm is used. For example, the value "10.5" gets rounded to "11", and the value "-10.5" gets rounded to "-10".



7.5.6.8 round-precision

Result is the decimal value of the number rounded to the decimal places defined by "decimals".

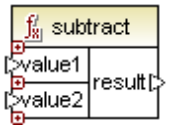




In the mapping above, the result is 0.429. For the result to appear correctly in an XML file, make sure to map it to an element of xs:decimal type.

7.5.6.9 subtract

Result is the decimal value of subtracting **value2** from **value1**.



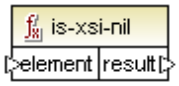
 © 2015 Altova GmbH

7.5.7 core | node functions

The node functions allow you to access nodes, or process nodes in a particular way.

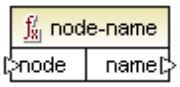
7.5.7.1 is-xsi-nil

Returns true (<OrderID>true</OrderID>) if the element node, of the source componenent, has the xsi:nil attribute set to "true".

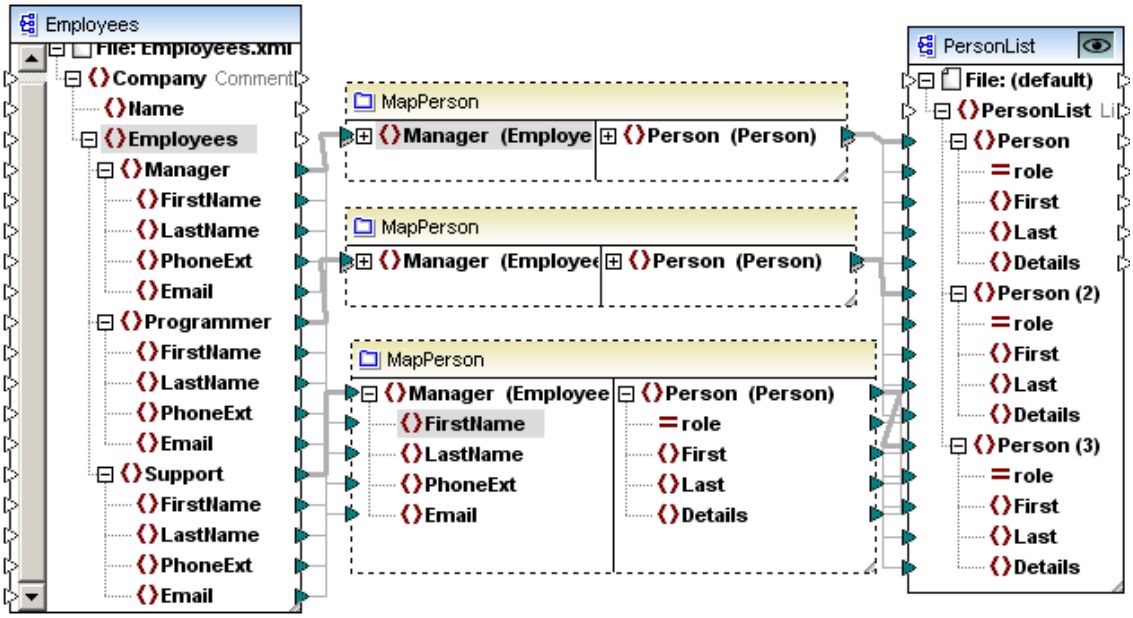


7.5.7.2 node-name

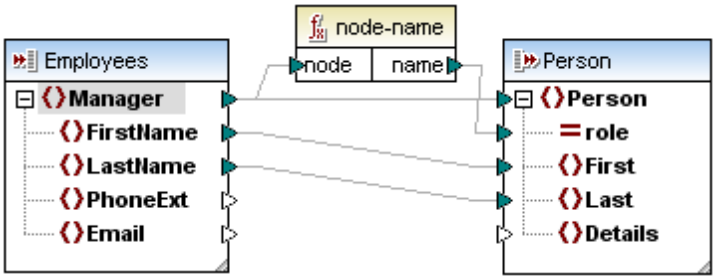
Returns the QName of the connected node unless it is an XML text() node; if this is the case, an empty QName is returned. This function only works on those nodes that have a name. If XSLT is the target language (which calls fn:node-name), it returns an empty sequence for nodes which have no names.



- Getting a name from database tables/fields is not supported.
- XBRL and Excel are not supported.
- Getting a name of File input node is not supported.
- WebService nodes behave like XML nodes except that:
  - node-name from "part" is not supported.
  - node-name from root node ("Output" or "Input") is not supported.



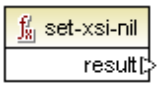
The MapPerson user-defined function uses **node-name** to return the name of the input **node**, and place it in the role attribute. The root node of the Employees.xsd, in the user-defined function, has been defined as "Manager".



Manager gets its data from **outside** the user-defined function, where it can be either: Manager, Programmer, or Support. This is the data that is then passed on to the role attribute in PersonList.

7.5.7.3 set-xsi-nil

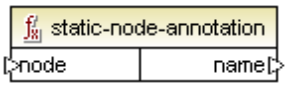
Sets the target node to xsi:nil.



7.5.7.4 static-node-annotation

Returns the string with annotation of the connected node. The input must be: (i) a source component node, or (ii) an inline function that is directly connected to a parameter, which in turn is directly connected to a node in the calling mapping.

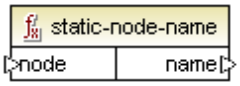
 © 2015 Altova GmbH



The connection must be direct. It cannot pass through a filter or a non-inlined user-defined function. This is a pseudo-function, which is replaced at generation time with the text acquired from the connected node, and is therefore available for all languages.

7.5.7.5 static-node-name

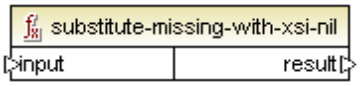
Returns the string with the name of the connected node. The input must be: (i) a source component node, or (ii) an inline function that is directly connected to a parameter, which in turn is directly connected to a node in the calling mapping.



The connection must be direct. It cannot pass through a filter or a non-inlined user-defined function. This is a pseudo-function, which is replaced at generation time with the text acquired from the connected node, and is therefore available for all languages.

7.5.7.6 substitute-missing-with-xsi-nil

For nodes with simple content, this function substitutes any missing (or null values) of the source component, with the `xsi:nil` attribute in the target node.



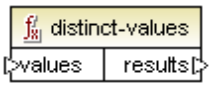
7.5.8 core | sequence functions

Sequence functions allow processing of input sequences and grouping of their content. The value/content of the **key** input parameter, mapped to nodes/rows, is used to group the sequence.

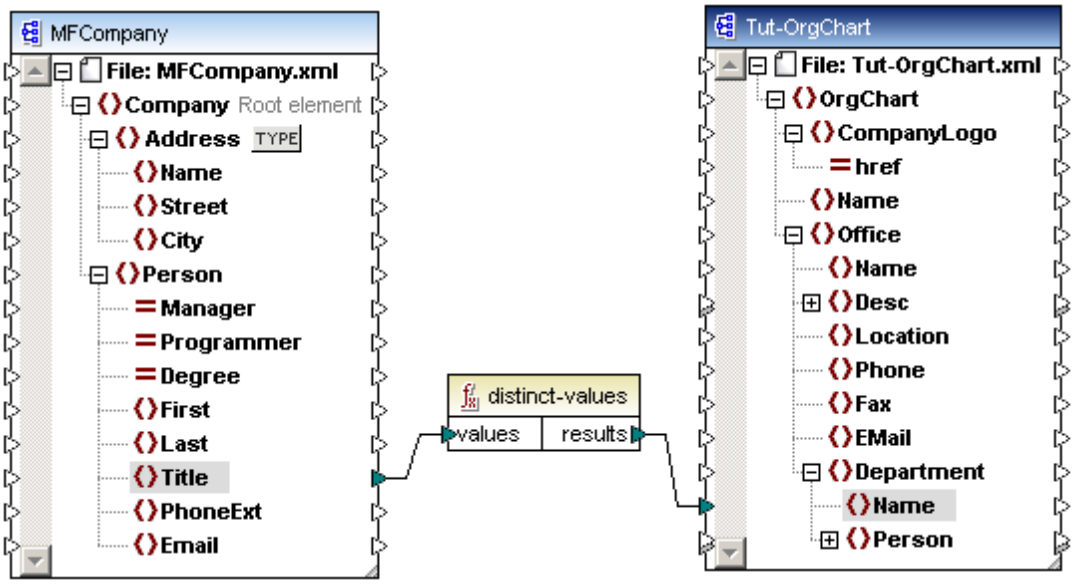
- Input parameter **key** is of an arbitrary data type that can be converted to string for **group-adjacent** and **group-by**
- Input parameter **bool** is of type Boolean for **group-starting-with** and **group-ending-with**
- The output **key** is the key of the current group.

7.5.8.1 distinct-values

Allows you to remove duplicate values from a sequence and map the unique items to the target component.



In the example below, the content of the source component "Title" items, are scanned and each unique title is mapped to the Department / Name item in the target component.

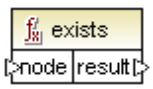


Note that the sequence of the individual Title items in the source component are retained when mapped to the target component.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <OrgChart xsi:schemaLocation="http://www.xmlspy.com/schemas/orgchart C:/DOCU
3    <Office>
4      <Department>
5        <Name>Office Manager</Name>
6        <Name>Accounts Receivable</Name>
7        <Name>Accounting Manager</Name>
8        <Name>Marketing Manager Europe</Name>
9        <Name>Art Director</Name>
10       <Name>Program Manager</Name>
11       <Name>Software Engineer</Name>
12       <Name>Technical Writer</Name>
13       <Name>IT Manager</Name>
14       <Name>Web Developer</Name>
15       <Name>Support Engineer</Name>
16       <Name>PR &amp;amp; Marketing Manager US</Name>
17     </Department>
18   </Office>
19 </OrgChart>
```

7.5.8.2 exists

Returns true if the node exists, else returns false.

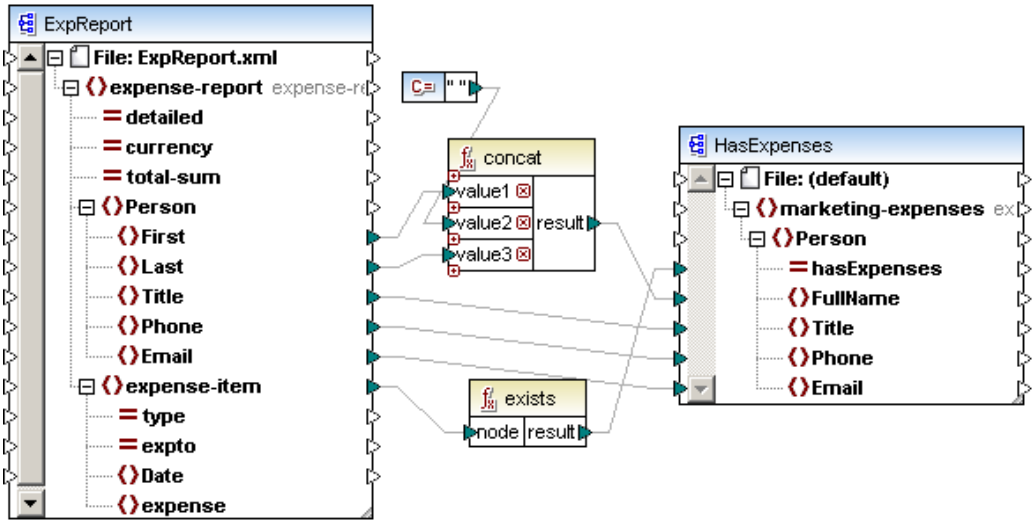


The "**HasMarketingExpenses.mfd**" file in the ...\MapForceExamples folder contains the small

 © 2015 Altova GmbH

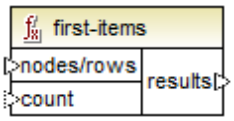
example shown below.

If an expense-item exists in the source XML, then the "hasExpenses" attribute is set to "true" in the target XML/Schema file.



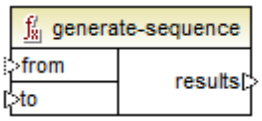
7.5.8.3 *first-items*

Returns the first "X" items of the input sequence, where X is the number supplied by the "count" parameter. E.g. if the value 3 is mapped to the count parameter and a parent node to the nodes/row parameter, then the first three items will be listed in the output.



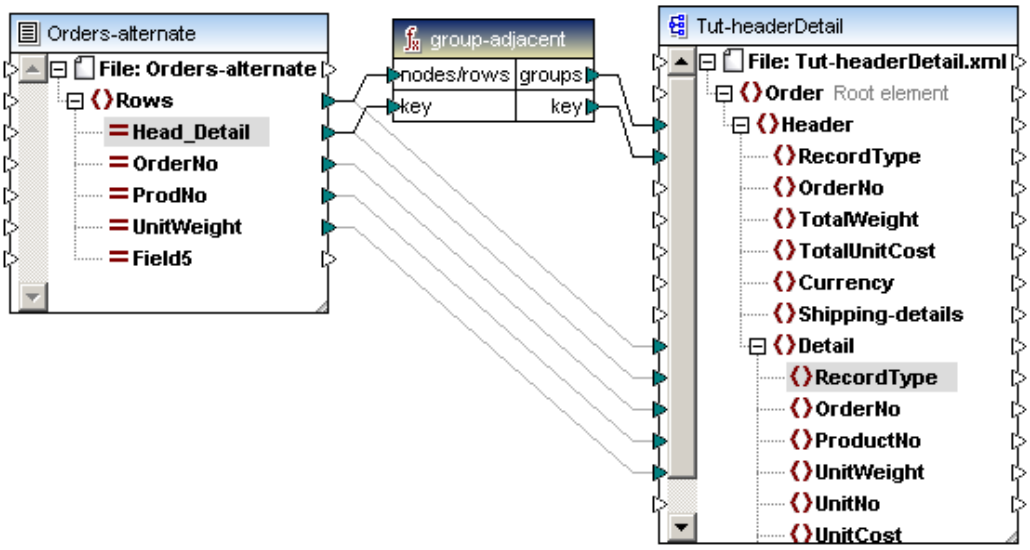
7.5.8.4 *generate-sequence*

Creates a sequence of integers using the "from" and "to" parameters as the boundaries.



7.5.8.5 *group-adjacent*

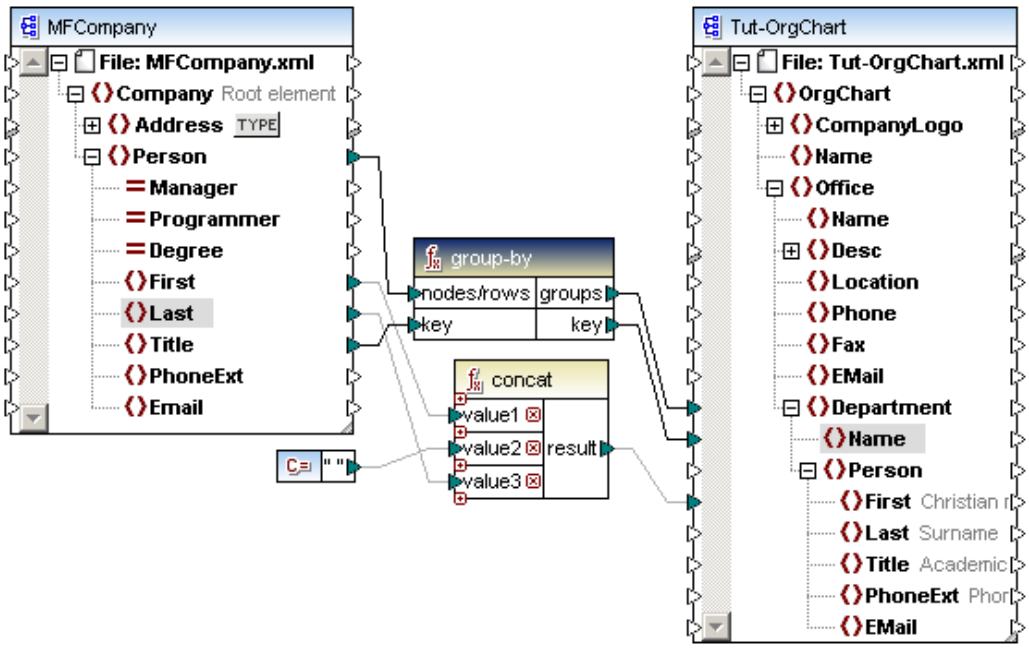
Groups the input sequence **nodes/rows** into groups of adjacent items sharing the same **key**. Note that group-adjacent uses the **content** of the node/item as the grouping key.



7.5.8.6 group-by

Groups the input sequence **nodes/rows** into groups of not necessarily adjacent items sharing the same **key**. Groups are output in the order the key occurs in the input sequence. The example below shows how this works:

- The key that defines the specific groups of the source component is the Title item. This is used to group the persons of the company.
- The group name is placed in the Department/Name item of the target component, while the concatenated person's first and last names are placed in the Person/First child item.



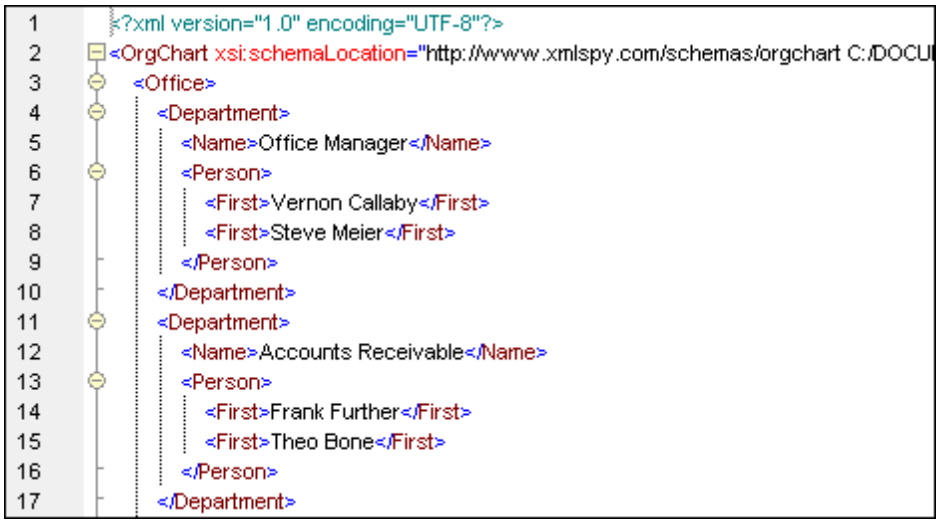
Note that group-by uses the **content** of the node/item as the grouping key. The content of the

 © 2015 Altova GmbH

Title field is used to group the persons and is mapped to the Department/Name item in the target.

Note also: there is an **implied filter** of the rows from the source document to the target document, which can be seen in the included example. In the target document, each Department item only has those Person items that **match** the grouping **key**, as the group-by component creates the necessary hierarchy on the fly.

Clicking the Output button shows the result of the grouping process.

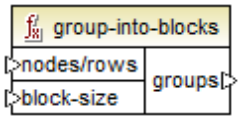


7.5.8.7 *group-ending-with*

This function groups the input sequence **nodes/rows** into groups, ending a new group whenever **bool** is true.

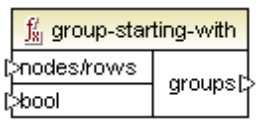
7.5.8.8 *group-into-blocks*

Groups the input sequence **nodes/rows** into blocks of the same size defined by the number supplied by the **block-size** parameter.

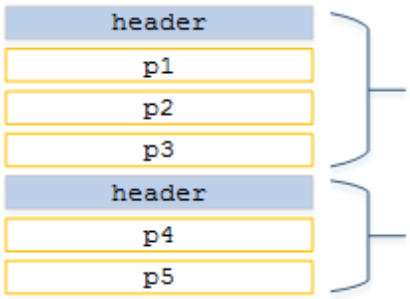


7.5.8.9 *group-starting-with*

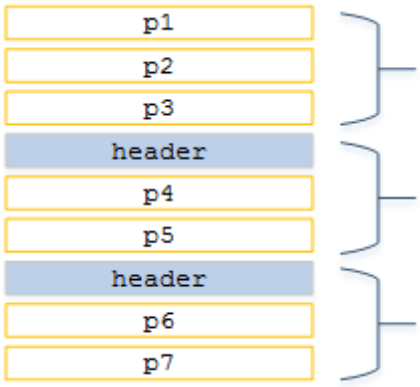
This function groups the input sequence **nodes/rows** into groups, starting a new group when **bool** is `true`.



The following example illustrates a sequence of nodes where **bool** returns `true` whenever the node "header" is encountered. Applying the **`group-starting-with`** function on this sequence of nodes results in two groups, as shown below.

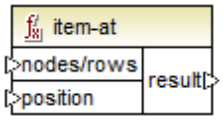


Note that the first node in the sequence starts a new group regardless of the value of **bool**. In other words, a sequence such as the one below would create three groups.



7.5.8.10 item-at

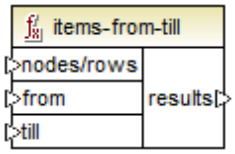
Returns the **nodes/rows** at the position supplied by the **position** parameter. The first item is at position "1".



7.5.8.11 items-from-till

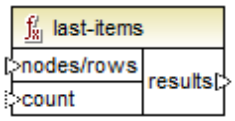
Returns a sequence of **nodes/rows** using the "from" and "till" parameters as the boundaries. The first item is at position "1".

 © 2015 Altova GmbH



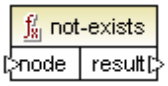
7.5.8.12 last-items

Returns the last "X" **nodes/rows** of the sequence where X is the number supplied by the "count" parameter. The first item is at position "1".



7.5.8.13 not-exists

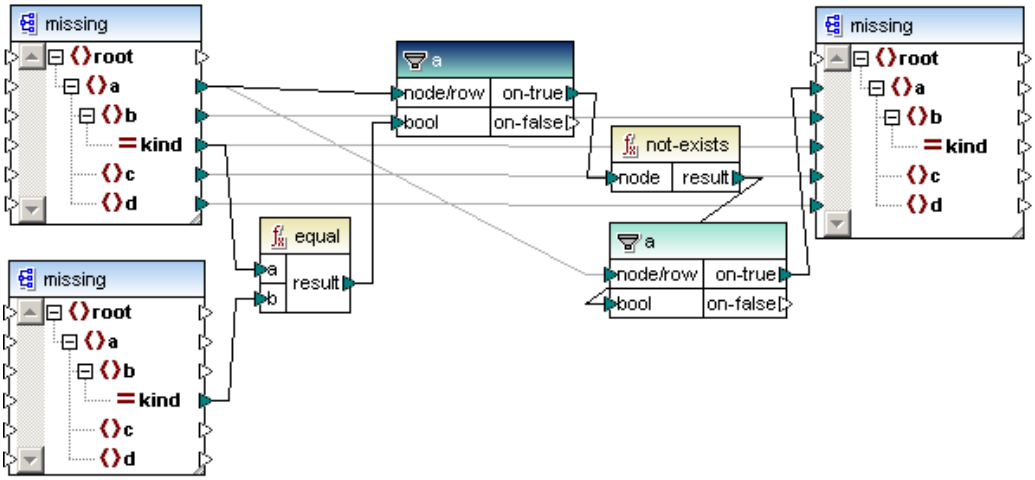
Returns false if the node exists, else returns true.



The example below shows how you can use the not-exists function to map nodes that do not exist in one of a pair of source files.

What this mapping does:

- Compare the nodes of two source XML files
- Filter out the nodes of the first source XML file, that do not exist in the second source XML file
- Map only the missing nodes, and their content, to the target file.

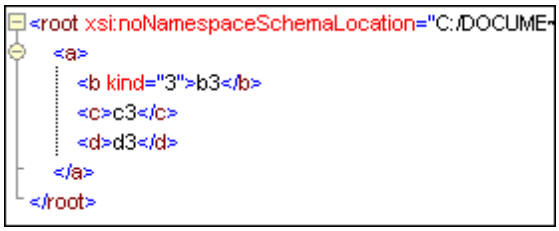


The two XML instance files are shown below, the differences between them are:

- **a.xml** (left) contains the node <b **kind="3"**>, which is missing from b.xml.
- **b.xml** (right) contains the node <b **kind="4">** which is missing from a.xml.

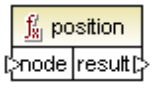
- The equal function compares the kind attribute of both XML files and passes the result to the filter.
- A not-exists function is placed after the initial filter, to select the missing nodes of each of the source files.
- The second filter is used to pass on the missing node and other data only from the a.xml file to the target.

The mapping result is that the node missing from b.xml, <b kind="3">, is passed on to the target component.



7.5.8.14 position

Returns the position of a node inside its containing sequence.

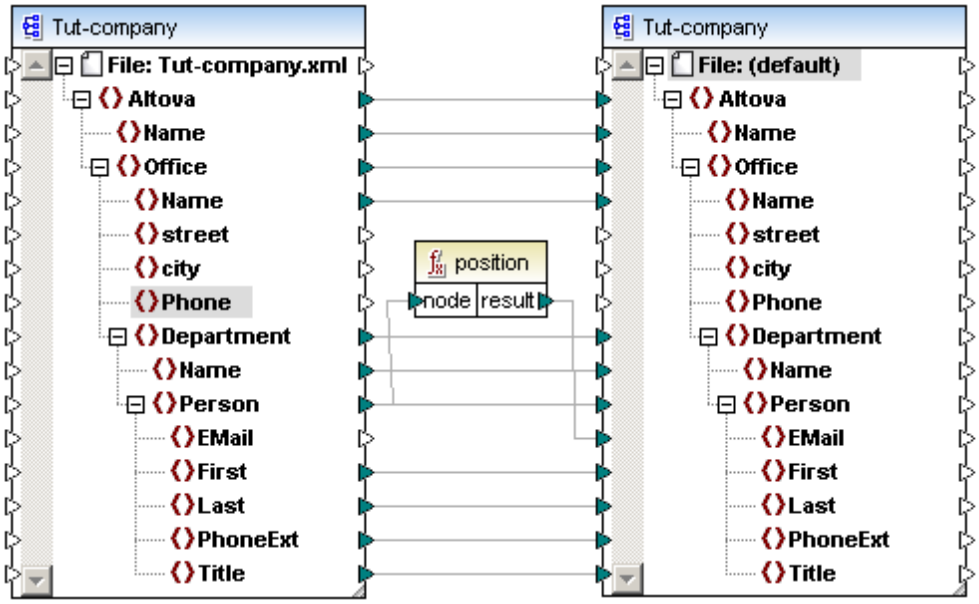


The position function allows you to determine the position of a specific node in a sequence, or use a specific position to filter out items based on that position.

The context item is defined by the item connected to the "node" parameter of the position function, Person, in the example below.

 © 2015 Altova GmbH

The simple mapping below adds a position number to each Person of each Department.



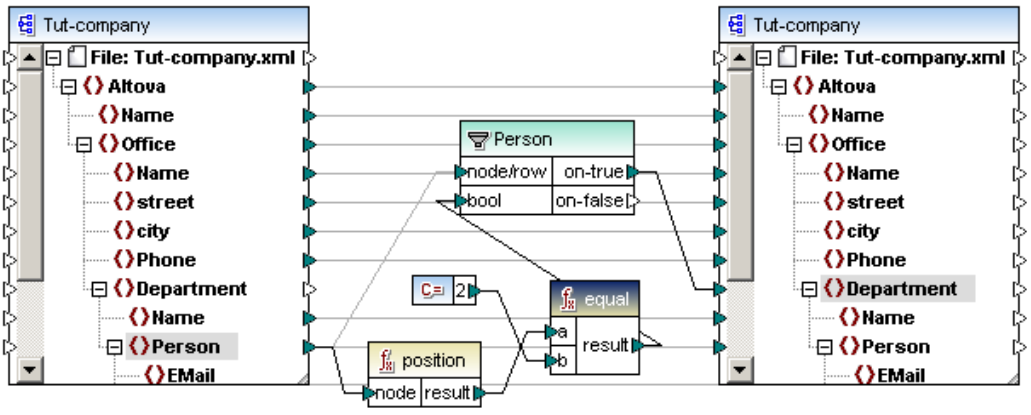
The position number is reset for each Department in the Office.

```
<Office>
  <Name>Microtech, Inc.</Name>
  <Department>
    <Name>Admin</Name>
    <Person>
      <EMail>1</EMail>
      <First>Albert</First>
      <Last>Aldrich</Last>
      <PhoneExt>582</PhoneExt>
      <Title>Manager</Title>
    </Person>
    <Person>
      <EMail>2</EMail>
      <First>Bert</First>
      <Last>Bander</Last>
      <PhoneExt>471</PhoneExt>
      <Title>Accounts Receivable</Title>
    </Person>
```

Using the position function to filter out specific nodes

Using the position function in conjunction with a filter allows you to map only those specific nodes that have a certain position in the source component.

The filter "node/row" parameter and the position "node" must be connected to the same item of the source component, to filter out a specific position of that sequence.



What this mapping does is to output:

- The second Person in each Department
- of each Office in Altova.

```
<Office>
  <Name>Microtech, Inc.</Name>
  <Department>
    <Name>Admin</Name>
    <Person>
      <EMail>b.bander@microtech.com</EMail>
      <First>Bert</First>
      <Last>Bander</Last>
      <Title>Accounts Receivable</Title>
    </Person>
  </Department>
  <Department>
    <Name>Sales and Marketing</Name>
    <Person>
      <EMail>e.ellas@microtech.com</EMail>
      <First>Eve</First>
      <Last>Ellas</Last>
      <Title>Art Director</Title>
    </Person>
  </Department>
  <Department>
    <Name>Manufacturing</Name>
    <Person>
      <EMail>g.gundall@microtech.com</EMail>
```

Finding the position of items in a filtered sequence:

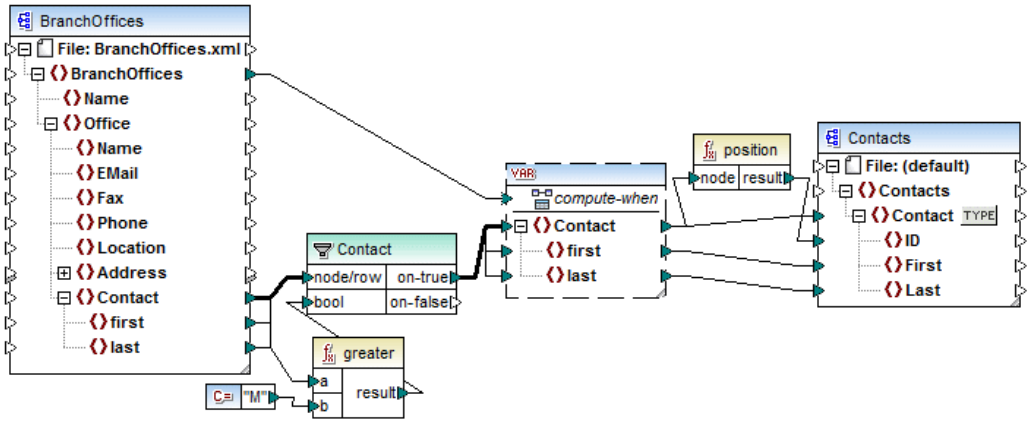
As the filter component is not a sequence function, it cannot be used directly in conjunction with the position function to find the position of filtered items. To do this you have to use the "Variable" component.

The results of a Variable component are always sequences, i.e. a delimited list of values, which can also be used to create sequences.

- The variable component is used to collect the filtered contacts where the last name starts with a letter higher than "M".
- The contacts are then passed on (from the variable) to the target component

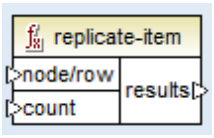
 © 2015 Altova GmbH

- The position function then numbers these contacts sequentially



7.5.8.15 *replicate-item*

Repeats every item in the input sequence the number of times specified in the `count` argument. If you connect a single item to the `node/row` argument, the function returns N items, where N is the value of the `count` argument. If you connect a sequence of items to the `node/row` argument, the function repeats each individual item in the sequence `count` times, processing one item at a time. For example, if `count` is 2, then the sequence `(1,2,3)` produces `(1,1,2,2,3,3)`.

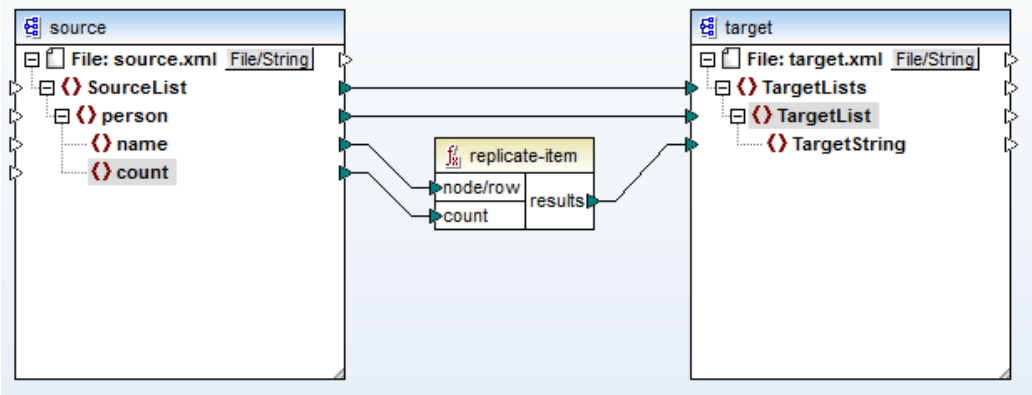


Note that you can supply a different `count` value for each item. For example, let's assume that you have a source XML file with the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<SourceList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="source.xsd">
   <person>
      <name>Michelle</name>
      <count>2</count>
   </person>
   <person>
      <name>Ted</name>
      <count>4</count>
   </person>
   <person>
      <name>Ann</name>
      <count>3</count>
   </person>
</SourceList>
```

With the help of the **`replicate-item`** function, you can repeat each person name a different number of times in a target component. To achieve this, connect the `<count>` node of each

person to the `count` input of the **`replicate-item`** function:

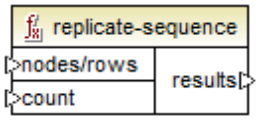


The output is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<TargetLists xsi:noNamespaceSchemaLocation="target.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <TargetList>
      <TargetString>Michelle</TargetString>
      <TargetString>Michelle</TargetString>
   </TargetList>
   <TargetList>
      <TargetString>Ted</TargetString>
      <TargetString>Ted</TargetString>
      <TargetString>Ted</TargetString>
      <TargetString>Ted</TargetString>
   </TargetList>
   <TargetList>
      <TargetString>Ann</TargetString>
      <TargetString>Ann</TargetString>
      <TargetString>Ann</TargetString>
   </TargetList>
</TargetLists>
```

7.5.8.16 replicate-sequence

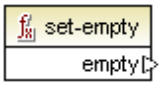
Repeats all items in the input sequence the number of times specified in the `count` argument. For example, if `count` is 2, then the sequence `(1,2,3)` produces `(1,2,3,1,2,3)`.



 © 2015 Altova GmbH

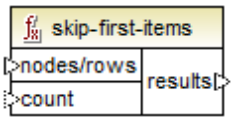
7.5.8.17 set-empty

Returns an empty sequence.



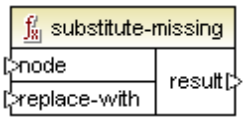
7.5.8.18 skip-first-items

Skips the first "X' items/nodes of the input sequence, where X is the number supplied by the "count" parameter, and returns the rest of the sequence.



7.5.8.19 substitute-missing

This function is a convenient combination of exists and a suitable if-else condition. Used to map the current field content if the node exists in the XML source file, otherwise use the item mapped to the "replace-with" parameter.

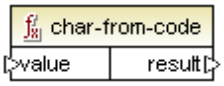


7.5.9 core | string functions

The string functions allow you to use the most common string functions to manipulate many types of source data to: extract portions, test for substrings, or retrieve information on strings.

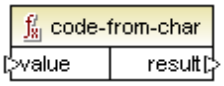
7.5.9.1 char-from-code

Result is the character representation of the decimal Unicode value of **value**.



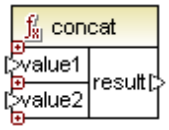
7.5.9.2 code-from-char

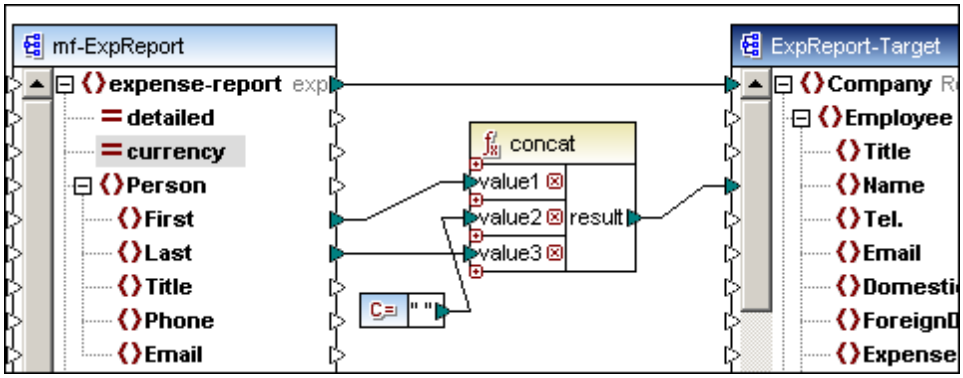
Result is the decimal Unicode value of the first character of **value**.



7.5.9.3 concat

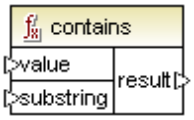
Concatenates (appends) two or more values into a single result string. All input values are auto matically converted to type string.





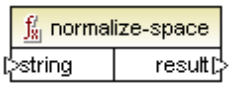
7.5.9.4 contains

Result is true if data supplied to the value parameter contains the string supplied by the substring parameter.



7.5.9.5 normalize-space

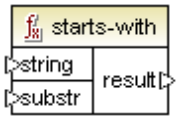
Result is the normalized input string, i.e. leading and trailing spaces are removed, then each sequence of multiple consecutive whitespace characters are replaced by a single whitespace character. The Unicode character for "space" is (U+0020).



 © 2015 Altova GmbH

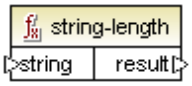
7.5.9.6 starts-with

Result is true if the input string "string" starts with **substr**, else false.



7.5.9.7 string-length

Result is the number of characters supplied by the **string** parameter.



7.5.9.8 substring

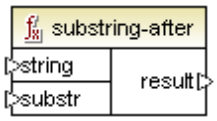
Result is the substring (string fragment) of the "string" parameter where "start" defines the position of the start character, and "length" the length of the substring.

If the length parameter is not specified, the result is a fragment starting at the start position and ending at the end position of the string. Indices start counting at 1.

E.g. substring("56789",2,3) results in 678.

7.5.9.9 substring-after

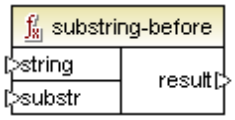
Result is the remainder of the "**string**" parameter, where the first occurrence of the **substr** parameter defines the start characters; the remainder of the string is the result of the function. An empty string is the result, if **substr** does not occur in **string**.



E.g. substring-after("2009/01/04","/") results in the substring 01/04. substr in this case is the first "/"character.

7.5.9.10 substring-before

Result is the string fragment of the "**string**" parameter, up to the first occurrence of the **substr** characters. An empty string is the result, if **substr** does not occur in **string**.



E.g. substring-before ("2009/01/04","/") results in the substring 2009. substr in this case is the first "**/**" character.

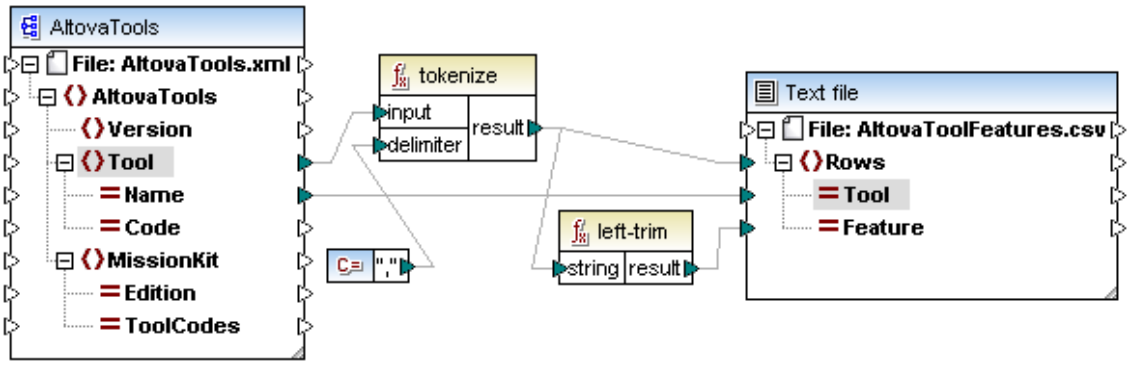
7.5.9.11 tokenize

Result is the input string split into a sequence of chunks/sections defined by the delimiter parameter. The result can then be passed on for further processing.

E.g. Input string is A,B,C and delimiter is "," - then result is A B C.

Example

The **tokenizeString1.mfd** file available in the ...\MapForceExamples folder shows how the tokenize function is used.



The XML source file is shown below. The **Tool** element has two attributes: Name and Code, with the Tool element data consisting of comma delimited text.

 © 2015 Altova GmbH

AltovaTools

xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xsi:noName...	AltovaTools.xsd
Version	2010

Tool (9)

	Name	Code	Text
1	XMLSpy	XS	XML editor, XSLT editor, XSLT debugger, XQuery editor, XQuery debugger,
2	MapForce	MF	Data integration, XML mapping, database mapping, text conversion, EDI tran
3	StyleVision	SV	Stylesheet designer, electronic forms, XSLT design, XSL:FO design, databa
4	UModel	UM	UML modeling tool, code generation, reverse engineering, UML, BPMN, Sysl
5	DatabaseSpy	DS	Multi-database tool, SQL auto-completion, graphical database design, table b
6	DiffDog	DD	Diff / merge tool, compare files, sync directories, compare XML, compare O
7	SchemaAgent	SA	XML Schema management tool, IIR management, XSLT management, WSDL
8	SemanticWorks	SW	Semantic Web tool, RDF editor, OWL editor, RDF/XML and N-Triples generati
9	Authentic	AU	XML authoring tool, database editor, XML publishing tool, e-Forms editor

MissionKit (4)

What the mapping does:

- The tokenize function receives data from the **Tool** element/item and uses the comma "**,**" delimiter to split that data into separate chunks. I.e. the first chunk "XML editor".
- As the result parameter is mapped to the **Rows** item in the target component, one **row** is generated for each chunk.
- The result parameter is also mapped to the **left-trim** function which removes the leading white space of each chunk.
- The result of the left-trim parameter (each chunk) is mapped to the **Feature** item of the target component.
- The target component output file has been defined as a CSV file (AltovaToolFeatures.csv) with the field delimiter being a semicolon (double click component to see settings).

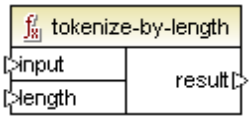
Result of the mapping:

- For each Tool element of the source file
- The (Tool) Name is mapped to the Tool item in the target component
- Each chunk of the tokenized Tool content is appended to the (Tool Name) Feature item
- E.g. The first tool, XMLSpy, gets the first Feature chunk "XML editor"
- This is repeated for all chunks of the current Tool and then for all Tools.
- Clicking the Output tab delivers the result shown below.

```
1   Tool;Feature
2   XMLSpy;XML editor
3   XMLSpy;XSLT editor
4   XMLSpy;XSLT debugger
5   XMLSpy;XQuery editor
6   XMLSpy;XQuery debugger
7   XMLSpy;XML Schema / DTD editor
8   XMLSpy;WSDL editor
9   XMLSpy;SOAP debugger
10  MapForce;Data integration
11  MapForce;XML mapping
12  MapForce;database mapping
```

7.5.9.12 tokenize-by-length

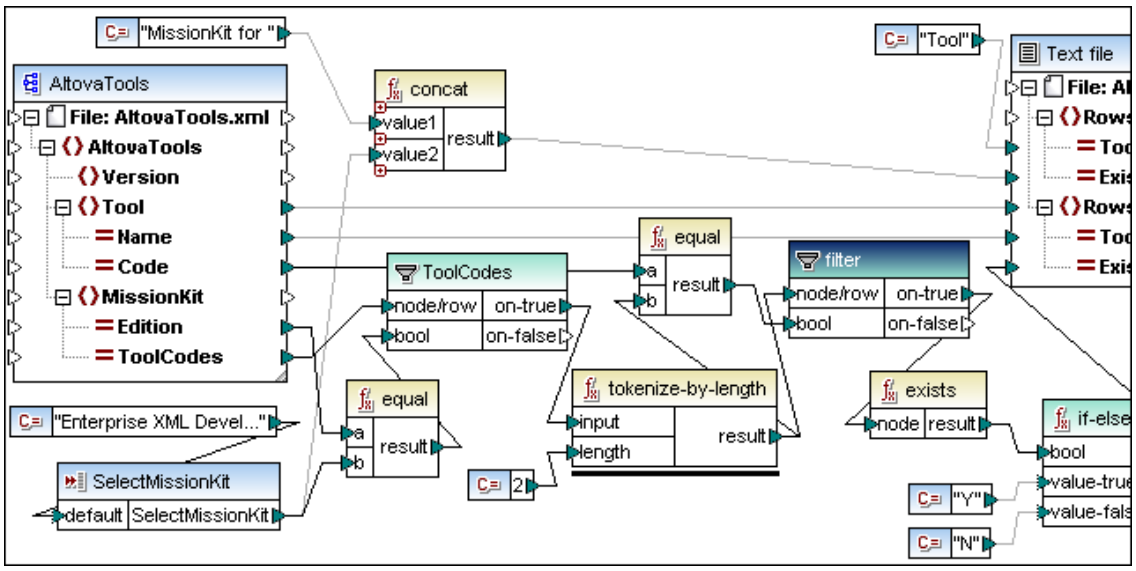
Result is the input string split into a sequence of chunks/sections defined by the length parameter. The result can then be passed on for further processing.



E.g. Input string is ABCDEF and length is "2" - then result is AB CD EF.

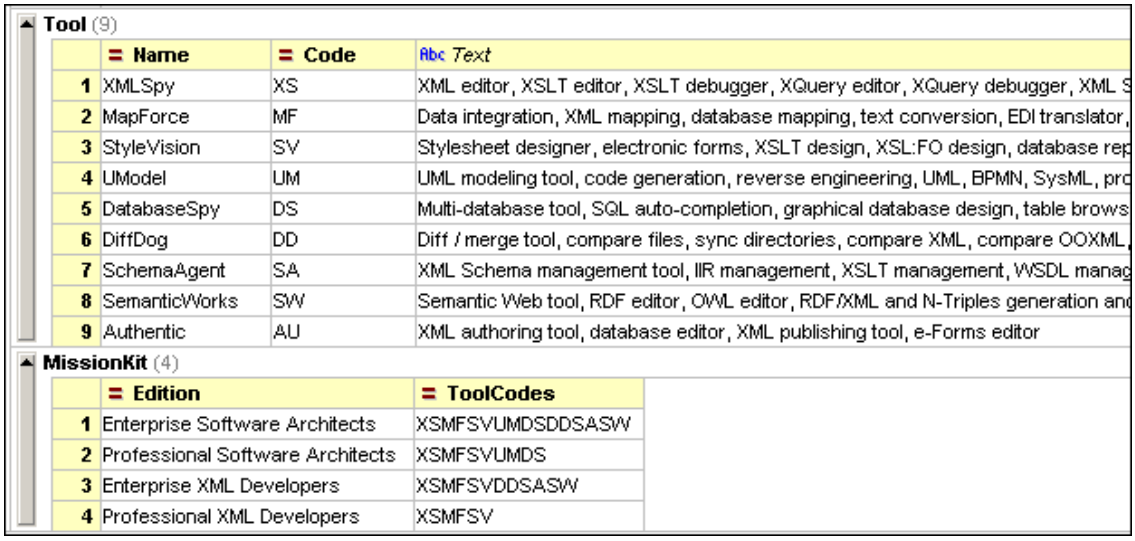
Example

The **tokenizeString2.mfd** file available in the ...\MapForceExamples folder shows how the **tokenize-by-length** function is used.



The XML source file is shown below, and is the same as the one used in the previous example. The **MissionKit** element also has two attributes: **Edition** and **ToolCodes**, but no MissionKit element content.

© 2015 Altova GmbH



Tool (9)

	= Name	= Code	Abc Text
1	XMLSpy	XS	XML editor, XSLT editor, XSLT debugger, XQuery editor, XQuery debugger, XML S
2	MapForce	MF	Data integration, XML mapping, database mapping, text conversion, EDI translator,
3	StyleVision	SV	Stylesheet designer, electronic forms, XSLT design, XSL:FO design, database rep
4	UModel	UM	UML modeling tool, code generation, reverse engineering, UML, BPMN, SysML, pro
5	DatabaseSpy	DS	Multi-database tool, SQL auto-completion, graphical database design, table brows
6	DiffDog	DD	Diff / merge tool, compare files, sync directories, compare XML, compare OOXML,
7	SchemaAgent	SA	XML Schema management tool, IIR management, XSLT management, WSDL manag
8	SemanticWorks	SW	Semantic Web tool, RDF editor, OWL editor, RDF/XML and N-Triples generation and
9	Authentic	AU	XML authoring tool, database editor, XML publishing tool, e-Forms editor

MissionKit (4)

	= Edition	= ToolCodes
1	Enterprise Software Architects	XSMFSVUMDSDDSASW
2	Professional Software Architects	XSMFSVUMDS
3	Enterprise XML Developers	XSMFSVDDSASW
4	Professional XML Developers	XSMFSV

Aim of the mapping:

To generate a list showing which Altova tools are part of the respective MissionKit editions.

How the mapping works:

- The SelectMissionKit **Input** component receives its default input from a constant component, in this case "Enterprise XML Developers".
- The **equal** function compares the input value with the "**Edition**" value and passes on the result to the **bool** parameter of the ToolCodes filter.
- The **node/row** input of the ToolCodes filter is supplied by the **ToolCodes** item of the source file. The value for the Enterprise XML Developers edition is: XSMFSVDDSASW.
- The XSMFSVDDSASW value is passed to the **on-true** parameter, and futher to the **input** parameter of the **tokenize-by-length** function.

What the tokenize-by-length function does:

- The ToolCodes **input** value XSMFSVDDSASW, is split into multiple chunks of two characters each, defined by **length** parameter, which is 2, thus giving 6 chunks.
- Each chunk (placed in the b parameter) of the equal function, is compared to the 2 character **Code** value of the source file (of which there are 9 entries/items in total).
- The result of the comparison (true/false) is passed on to the **bool** parameter of the filter.
- Note that **all** chunks, of the tokenize-by-length function, are passed on to the **node/row** parameter of the filter.

- The **exists** functions now checks for existing/non-existing nodes passed on to it by the **on-true** parameter of the filter component.

  Existing nodes are those where there **is** a match between the **ToolCodes** chunk and the Code value.

  Non-existing nodes are where there was **no ToolCodes** chunk to match a Code value.

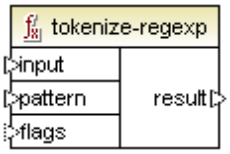
- The bool results of the **exists** function are passed on to the **if-else** function which passes on a Y to the target if the node exists, or a N, if the node does not exist.

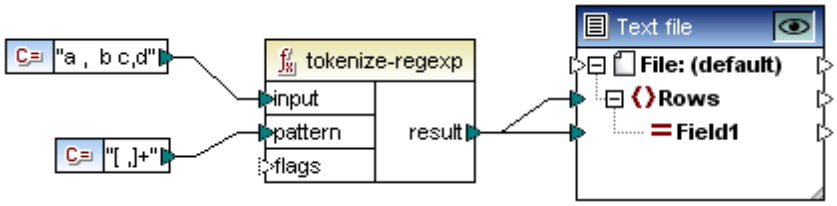
Result of the mapping:

```
1   Tool;MissionKit for Enterprise XML Developers
2   XMLSpy;Y
3   MapForce;Y
4   StyleVision;Y
5   UModel;N
6   DatabaseSpy;N
7   DiffDog;Y
8   SchemaAgent;Y
9   SemanticWorks;Y
10  Authentic;N
11
```

7.5.9.13 tokenize-regexp

Result is the input string split into a sequence of strings, where the supplied regular expression **pattern** match defines the separator. The separator strings are not output by the result parameter. Optional flags may also be used.





In the example shown above:
input string is a succession of characters separated by spaces and/or commas, i.e. a , b c,d

The regex **pattern** defines a character class **[**"space""comma"**]** - of which one and only one character will be matched in a character class, i.e. either space or comma.

The **+** quantifier specifies "one or more" occurrences of the character class/string.

result string is:

```
1   a
2   b
3   c
4   d
5
```

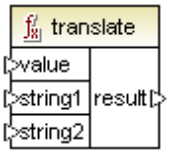
Please note that there are slight differences in regular expression syntax between the various languages. Tokenize-regexp in C++ is only available in Visual Studio 2008 SP1 and later.

For more information on regular expressions, see Regular expressions.

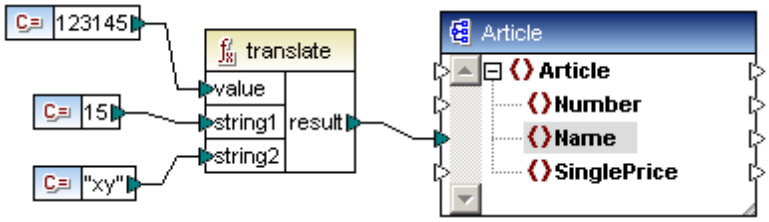
 © 2015 Altova GmbH

7.5.9.14 translate

The characters of **string1** (search string) are replaced by the characters at the same position in **string2 (replace string**), in the input string "**value"**.



When there are no corresponding characters in string2, the character is removed.



E.g.
input string is 123145
(search) string1 is 15
(replace) string2 is xy

So:
each 1 is replaced by **x** in the input string value
each 5 is replaced by **y** in the input sting value

Result string is **x23x4y**

If string2 is empty (fewer characters than string1) then the character is removed.

E.g.2
input string aabaacbca
string1 is "**a**"
string2 is **""** (empty string)

result string is "bcbc"

E.g.3
input string aabaacbca
string1 is "**ac**"
string2 is "**ca**"

result string is "ccbccabac"

7.5.10 xpath2 | accessors

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

7.5.10.1 base-uri

The `base-uri` function takes a node argument as input, and returns the URI of the XML resource containing the node. The output is of type `xs:string`. MapForce returns an error if no input node is supplied.

7.5.10.2 node-name

The `node-name` function takes a node as its input argument and returns its QName. When the QName is represented as a string, it takes the form of `prefix:localname` if the node has a prefix, or `localname` if the node has no prefix. To obtain the namespace URI of a node, use the `namespace-URI-from-QName` function (in the library of QName-related functions).

7.5.10.3 string

The `string` function works like the `xs:string` constructor: it converts its argument to `xs:string`.

When the input argument is a value of an atomic type (for example `xs:decimal`), this atomic value is converted to a value of `xs:string` type. If the input argument is a node, the string value of the node is extracted. (The string value of a node is a concatenation of the values of the node's descendant nodes.)

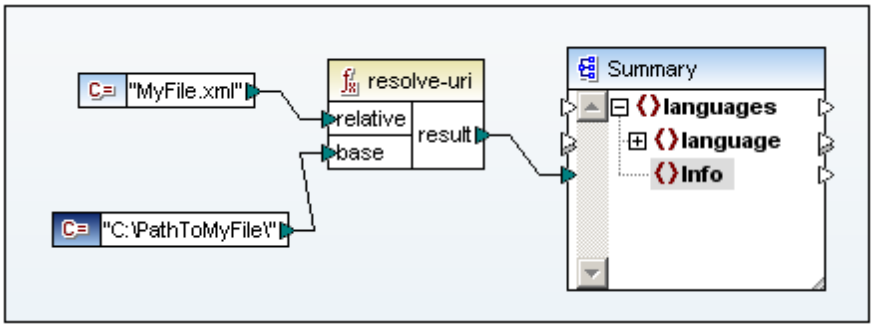
7.5.11 xpath2 | anyURI functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

7.5.11.1 resolve-uri

The `resolve-uri` function takes a URI as its first argument (datatype `xs:string`) and resolves it against the URI in the second argument (datatype `xs:string`).

The result (datatype `xs:string`) is a combined URI. In this way a relative URI (the first argument) can be converted to an absolute URI by resolving it against a base URI.



In the screenshot above, the first argument provides the relative URI, the second argument the

 © 2015 Altova GmbH

base URI. The resolved URI will be a concatenation of base URI and relative URI, so `C:\PathtoMyFile\MyFile.xml`.

Note: Both arguments are of datatype `xs:string` and the process of combining is done by treating both inputs as strings. So there is no way of checking whether the resources identified by these URIs actually exist. MapForce returns an error if the second argument is not supplied.

7.5.12 xpath2 | boolean functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.
The Boolean functions `true` and `false` take no argument and return the boolean constant values, `true` and `false`, respectively. They can be used where a constant boolean value is required.

7.5.12.1 false

Returns the Boolean value "false".

7.5.12.2 true

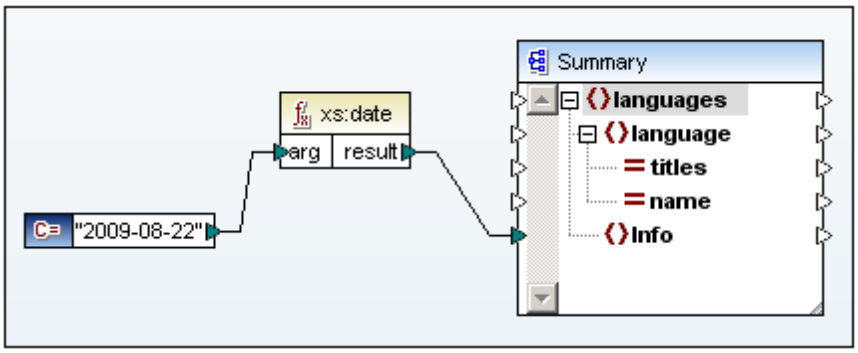
Returns the Boolean value "true".

7.5.13 xpath2 | constructors

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

The functions in the Constructors part of the XPath 2.0 functions library construct specific datatypes from the input text. Typically, the lexical format of the input text must be that expected of the datatype to be constructed. Otherwise, the transformation will not be successful.

For example, if you wish to construct an `xs:date` datatype, use the `xs:date` constructor function. The input text must have the lexical format of the `xs:date` datatype, which is: `YYYY-MM-DD` (*screenshot below*).



In the screenshot above, a string constant (`2009-08-22`) has been used to provide the input argument of the function. The input could also have been obtained from a node in the source

document.

The xs:date function returns the input text (2009-08-22), which is of xs:string datatype (specified in the *Constant* component), as output of xs:date datatype.

When you mouseover the input argument in a function box, the expected datatype of the argument is displayed in a popup.

7.5.14 xpath2 | context functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

The Context functions library contains functions that provide the current date and time, the default collation used by the processor, and the size of the current sequence and the position of the current node.

7.5.14.1 current-date

Returns the current date (xs:date) from the system clock.

7.5.14.2 current-dateTime

Returns the current date and time (xs:dateTime) from the system clock.

7.5.14.3 current-time

Returns the current time (xs:time) from the system clock.

7.5.14.4 default-collation

The default-collation function takes no argument and returns the default collation, that is, the collation that is used when no collation is specified for a function where one can be specified.

The Altova XSLT 2.0 Engine supports the Unicode codepoint collation only. Comparisons, including for the fn:max and fn:min functions, are based on this collation.

7.5.14.5 implicit-timezone

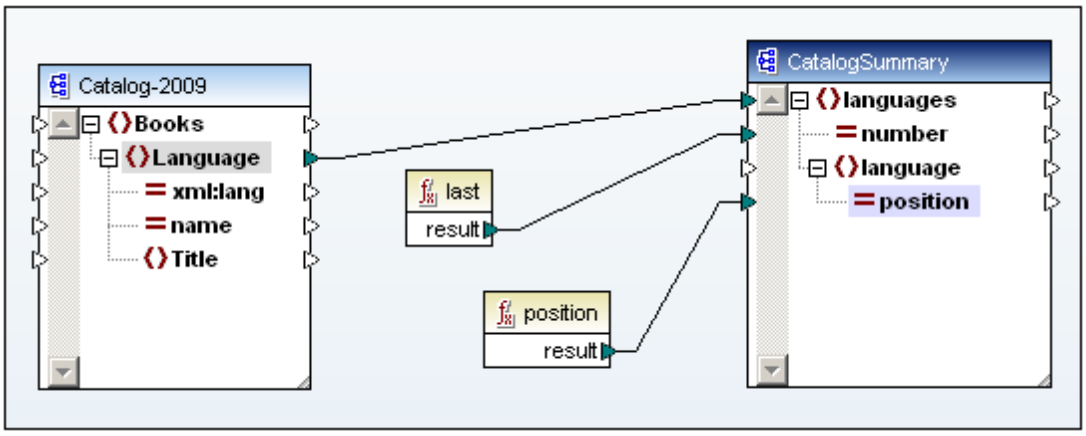
Returns the value of the "implicit timezone" property from the evaluation context.

7.5.14.6 last

The last and position functions take no argument. The last function returns the position of the last node in the context nodeset. The position function returns the position of the current node in the nodeset being processed.

 © 2015 Altova GmbH

The context nodeset at the nodes where the functions are directed, is the nodeset to which the functions will apply. In the screenshot below, the nodeset of `Language` elements is the context nodeset for the `last` and `position` functions.



In the example above, the `last` function returns the position of the last node of the context nodeset (the nodeset of `Language` elements) as the value of the `number` attribute. This value is also the size of the nodeset since it indicates the number of nodes in the nodeset.

The `position` function returns the position of the `Language` node being currently processed. For each `Language` element node, its position within the nodeset of `Langauge` elements is output to the `language/@position` attribute node.

We would advise you to use the **position** and **count** functions from the **core** library.

7.5.15 xpath2 | durations, date and time functions

XPath2 functions are available when either the XSLT2 or XQuery languages are selected.

The XPath 2 duration and date and time functions enable you to adjust dates and times for the timezone, extract particular components from date-time data, and subtract one date-time unit from another.

The 'Adjust-to-Timezone' functions

Each of these related functions takes a date, time, or dateTime as the first argument and adjusts the input by adding, removing, or modifying the timezone component depending on the value of the second argument.

The following situations are possible when the first argument contains no timezone (for example, the date `2009-01` or the time `14:00:00`).

- Timezone argument (the second argument of the function) is present: The result will contain the timezone specified in the second argument. The timezone in the second argument is added.
- Timezone argument (the second argument of the function) is absent: The result will contain the implicit timezone, which is the system's timezone. The system's timezone is added.
- Timezone argument (the second argument of the function) is empty: The result will

contain no timezone.

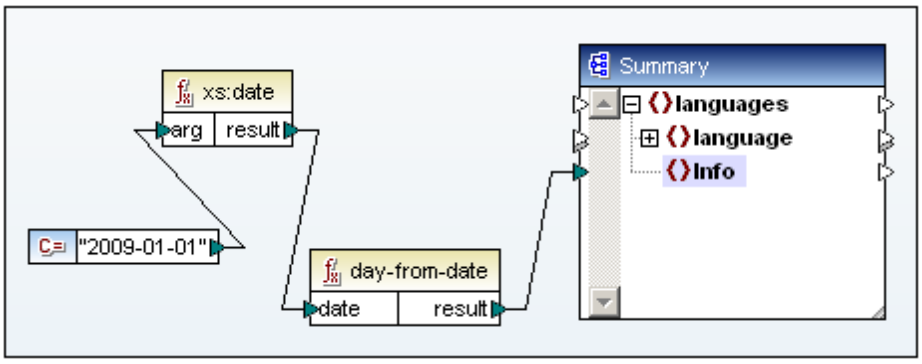
The following situations are possible when the first argument contains a timezone (for example, the date `2009-01-01+01:00` or the time `14:00:00+01:00`).

- Timezone argument (the second argument of the function) is present: The result will contain the timezone specified in the second argument. The original timezone is replaced by the timezone in the second argument.
- Timezone argument (the second argument of the function) is absent: The result will contain the implicit timezone, which is the system's timezone. The original timezone is replaced by the system's timezone.
- Timezone argument (the second argument of the function) is empty: The result will contain no timezone.

The 'From' functions

Each of the 'From' functions extracts a particular component from: (i) date or time data, and (ii) duration data. The results are of the xs:decimal datatype.

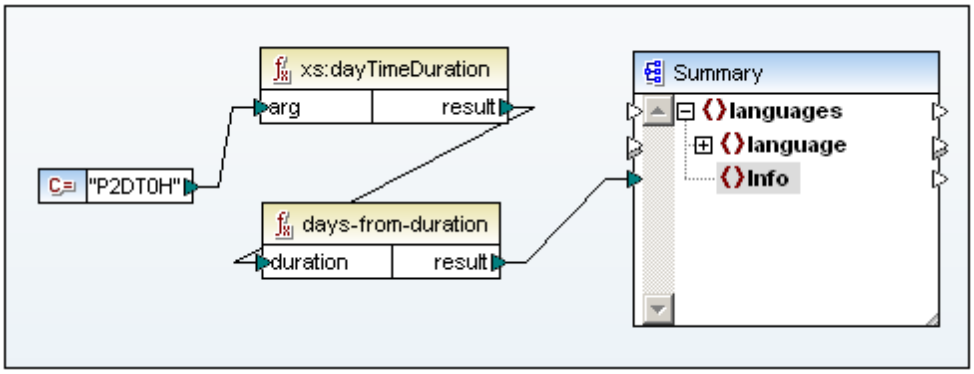
As an example of extracting a component from date or time data, consider the `day-from-date` function (*screenshot below*).



The input argument is a date (`2009-01-01`) of type `xs:date`. The `day-from-date` function extracts the day component of the date (`1`) as an xs:decimal datatype.

Extraction of time components from durations requires that the duration be specified either as `xs:yearMonthDuration` (for extracting years and months) or `xs:dayTimeDuration` (for extracting days, hours, minutes, and seconds). The result will be of type `xs:decimal`. The screenshot below shows a `dayTimeDuration` of `P2DT0H` being input to the `days-from-duration` function. The result is the xs:decimal `2`.

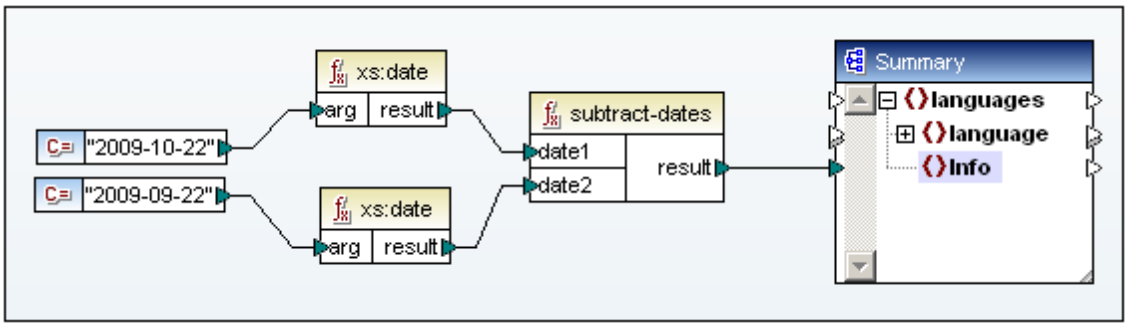
 © 2015 Altova GmbH



The 'Subtract' functions

Each of the three subtraction functions enables you to subtract one time value from another and return a duration value. The three subtraction functions are: `subtract-dates`, `subtract-times`, `subtract-dateTimes`.

The screenshot below shows how the subtract-dates function is used to subtract two dates (`2009-10-22` minus `2009-09-22`). The result is the dayTimeDuration `P30D`.

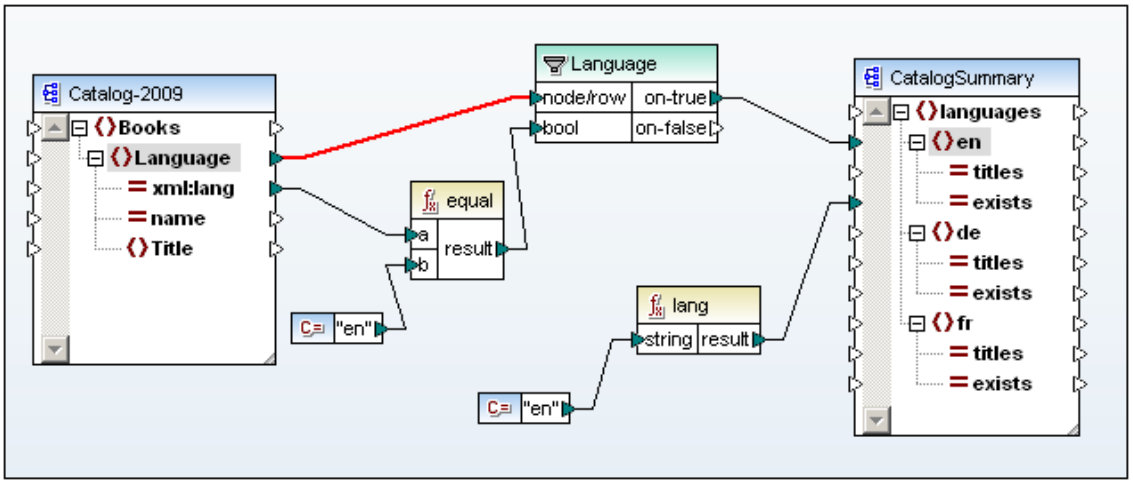


7.5.16 xpath2 | node functions

The following XPath 2 node functions are available:

lang

The `lang` function takes a string argument that identifies a language code (such as `en`). The function returns `true` or `false` depending on whether the context node has an `xml:lang` attribute with a value that matches the argument of the function.



In the screenshot above notice the following:

1. In the source schema, the `Language` element has an `xml:lang` attribute.
2. `Language` nodes are filtered so that only those `Language` nodes having an `xml:lang` value of `en` are processed (the filter test is specified in the `equal` function).
3. The `Language` node is the context node at the point where the `en` element is created in the output document.
4. The output of the `lang` function (`true` or `false`) is sent to the `en/@exists` attribute node of the output. The argument of the function is provided by the string constant `en`. The `lang` function then checks whether the context node at this point (the `Language` element) has an `xml:lang` attribute with a value of `en` (the argument of the function). If yes, then `true` is returned, otherwise `false`.

local-name, name, namespace-uri

The `local-name`, `name`, and `namespace-uri` functions, return, respectively, the local-name, name, and namespace URI of the input node. For example, for the node `altova:Products`, the local-name is `Products`, the name is `altova:Products`, and the namespace URI is the URI of the namespace to which the `altova:` prefix is bound (say, `http://www.altova.com/examples`).

Each of these three functions has two variants:

- With no argument: the function is then applied to the context node (for an example of a context node, see the example given for the `lang` function above).
- An argument that must be a node: the function is applied to the submitted node.

The output of each of these six variants is a string.

number

Converts an input string into a number. Also converts a boolean input to a number.

The number function takes a node as input, atomizes the node (that is, extracts its contents), and converts the value to a decimal and returns the converted value. The only types that can be converted to numbers are booleans, strings, and other numeric types. Non-numeric input values (such as a non-numeric string) result in `NaN` (Not a Number).

There are two variants of the number function:

 © 2015 Altova GmbH

- With no argument: the function is then applied to the context node (for an example of a context node, see the example given for the `lang` function above).
- An argument that must be a node: the function is applied to the submitted node.

7.5.17 xpath2 | numeric functions

The following XPath 2 numeric functions are available:

abs

The `abs` function takes a numeric value as input and returns its absolute value as a decimal. For example, if the input argument is `-2` or `+2`, the function returns `2`.

round-half-to-even

The `round-half-to-even` function rounds the supplied number (first argument) to the degree of precision (number of decimal places) supplied in the optional second argument. For example, if the first argument is `2.141567` and the second argument is `3`, then the first argument (the number) is rounded to three decimal places, so the result will be `2.141`. If no precision (second argument) is supplied, the number is rounded to zero decimal places, that is, to an integer.

The 'even' in the name of the function refers to the rounding to an even number when a digit in the supplied number is midway between two values. For example, `round-half-to-even(3.475, 2)` would return `3.48`.

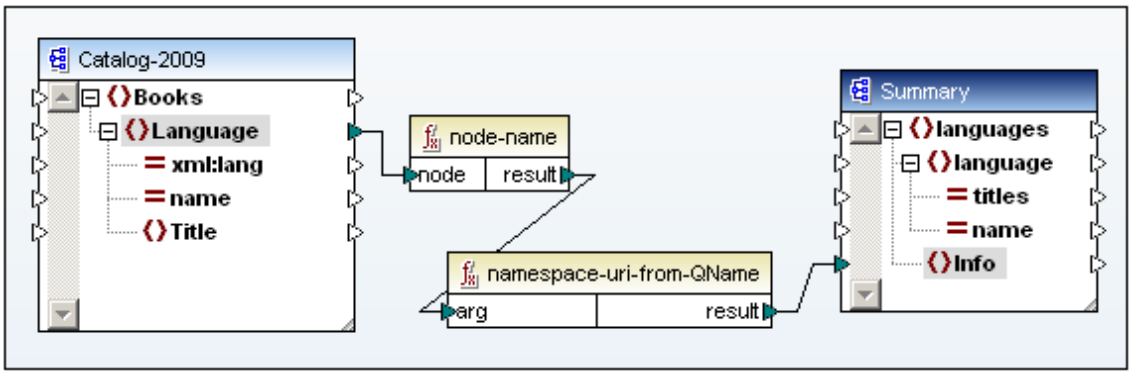
7.5.18 xpath2 | qname-related functions

There are two QName-related functions that work similarly: `local-name-from-QName` and `namespace-uri-from-QName`.

Both functions take an expanded QName (in the form of a string) as their input arguments and output, respectively, the local-name and namespace-uri part of the expanded QName.

The important point to note is that since the input of both functions are strings, a node cannot be connected directly to the input argument boxes of these functions.

The node should first be supplied to the `node-name` function, which outputs the expanded QName. This expanded QName can then be provided as the input to the two functions (*see screenshot below*).



The output of both functions is a string.

7.5.19 xpath2 | string functions

The following XPath 2 string functions are available:

compare

The `compare` function takes two strings as arguments and compares them for equality and alphabetically. If *String-1* is alphabetically less than *String-2* (for example the two string are: `A` and `B`), then the function returns `-1`. If the two strings are equal (for example, `A` and `A`), the function returns `0`. If *String-1* is greater than *String-2* (for example, `B` and `A`), then the function returns +1.

A variant of this function allows you to choose what collation is to be used to compare the strings. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

ends-with

The `ends-with` function tests whether *String-1* ends with *String-2*. If yes, the function returns `true`, otherwise `false`.

A variant of this function allows you to choose what collation is to be used to compare the strings. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

escape-uri

The `escape-uri` function takes a URI as input for the first string argument and applies the URI escaping conventions of RFC 2396 to the string. The second boolean argument (`escape-reserved`) should be set to `true()` if characters with a reserved meaning in URIs are to be escaped (for example "+" or "/").

For example:

`escape-uri("My A+B.doc", true())` would give `My%20A%2B.doc`
`escape-uri("My A+B.doc", false())` would give `My%20A+B.doc`

lower-case

The `lower-case` function takes a string as its argument and converts every upper-case character in the string to its corresponding lower-case character.

matches

The `matches` function tests whether a supplied string (the first argument) matches a regular expression (the second argument). The syntax of regular expressions must be that defined for the `pattern` facet of XML Schema. The function returns `true` if the string matches the regular expression, `false` otherwise.

The function takes an optional `flags` argument. Four flags are defined (`i`, `m`, `s`, `x`). Multiple flags

 © 2015 Altova GmbH

can be used: for example, `imx`. If no flag is used, the default values of all four flags are used.

The meaning of the four flags are as follows:

- `i` Use case-insensitive mode. The default is case-sensitive.
- `m` Use multiline mode, in which the input string is considered to have multiple lines, each separated by a newline character (`x0a`). The meta characters `^` and `$` indicate the beginning and end of each line. The default is string mode, in which the string starts and ends with the meta characters `^` and `$`.
- `s` Use dot-all mode. The default is not-dot-all mode, in which the meta character `"."` matches all characters except the newline character (`x0a`). In dot-all mode, the dot also matches the newline character.
- `x` Ignore whitespace. By default whitespace characters are not ignored.

normalize-unicode

The `normalize-unicode` function normalizes the input string (the first argument) according to the rules of the normalization form specified (the second argument). The normalization forms NFC, NFD, NFKC, and NFKD are supported.

replace

The `replace` function takes the string supplied in the first argument as input, looks for matches as specified in a regular expression (the second argument), and replaces the matches with the string in the third argument.

The rules for matching are as specified for the matches attribute above. The function also takes an optional `flags` argument. The flags are as described in the `matches` function above.

starts-with

The `starts-with` function tests whether *String-1* starts with *String-2*. If yes, the function returns `true`, otherwise `false`.

A variant of this function allows you to choose what collation is to be used to compare the strings. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

substring-after

The substring-after function returns that part of *String-1* (the first argument) that occurs after the test string, *String-2* (the second argument). An optional third argument specifies the collation to use for the string comparison. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

substring-before

The substring-before function returns that part of *String-1* (the first argument) that occurs before the test string, *String-2* (the second argument). An optional third argument specifies the collation

to use for the string comparison. When no collation is used, the default collation, which is the Unicode codepoint collation, is used. The Altova Engines support the Unicode codepoint collation only.

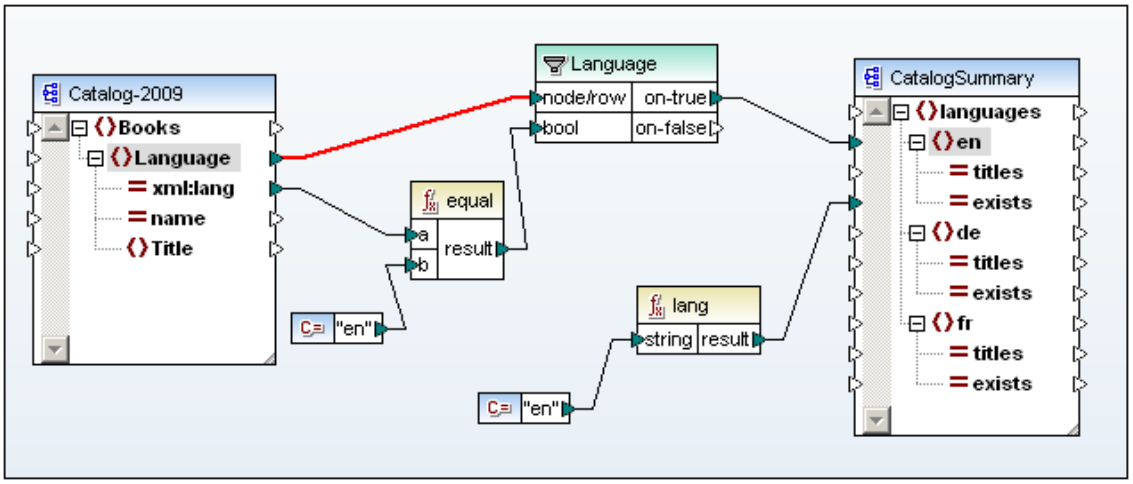
upper-case

The `upper-case` function takes a string as its argument and converts every lower-case character in the string to its corresponding upper-case character.

7.5.20 xslt | xpath functions

The functions in the XPath Functions library are XPath 1.0 nodeset functions. Each of these functions takes a node or nodeset as its context and returns information about that node or nodeset. These function typically have:

- a context node (in the screenshot below, the context node for the `lang` function is the Language element of the source schema).
- an input argument (in the screenshot below, the input argument for the `lang` function is the string constant `en`). The `last` and `position` functions take no argument.



lang

The `lang` function takes a string argument that identifies a language code (such as `en`). The function returns `true` or `false` depending on whether the context node has an `xml:lang` attribute with a value that matches the argument of the function. In the screenshot above notice the following:

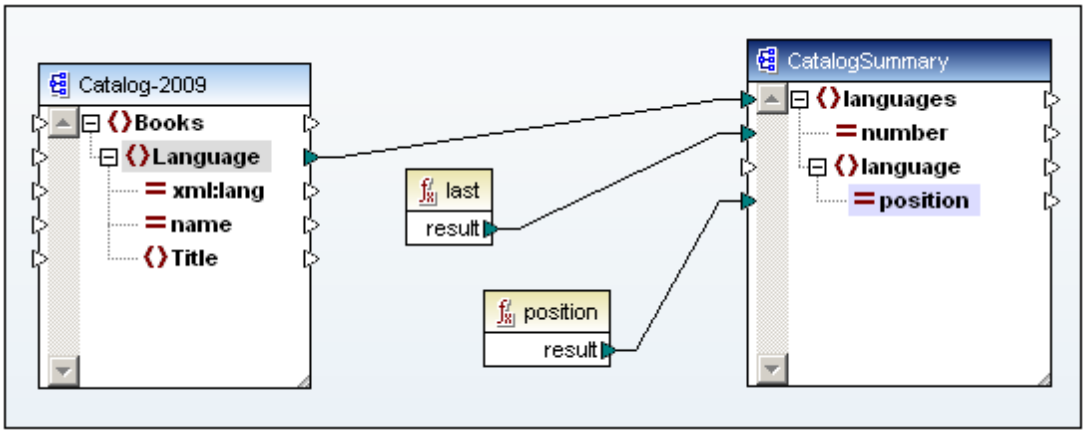
1. In the source schema, the `Language` element has an `xml:lang` attribute.
2. `Language` nodes are filtered so that only those `Language` nodes having an `xml:lang` value of `en` are processed (the filter test is specified in the `equal` function).
3. The `Language` node is the context node at the point where the `en` element is created in the output document.
4. The output of the `lang` function (`true` or `false`) is sent to the `en/@exists` attribute node of the output. The argument of the function is provided by the string constant `en`. The `lang` function then checks whether the context node at this point (the `Language` element) has an `xml:lang` attribute with a value of `en` (the argument of the function). If yes, then `true` is returned, otherwise `false`.

 © 2015 Altova GmbH

last, position

The `last` and `position` functions take no argument. The `last` function returns the position of the last node in the context nodeset. The `position` function returns the position of the current node in the nodeset being processed.

The context nodeset at the nodes where the functions are directed is the nodeset to which the functions will apply. In the screenshot below, the nodeset of `Language` elements is the context nodeset for the `last` and `position` functions.



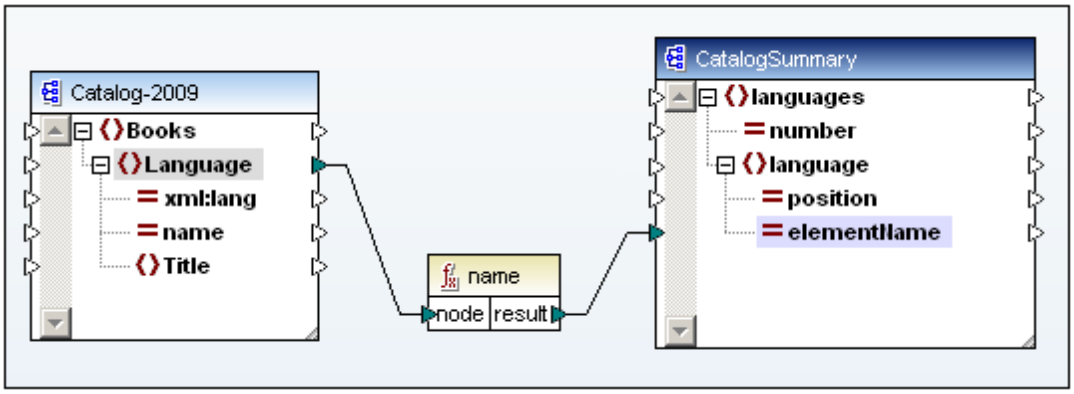
In the example above, the `last` function returns the position of the last node of the context nodeset (the nodeset of `Language` elements) as the value of the `number` attribute. This value is also the size of the nodeset since it indicates the number of nodes in the nodeset.

The `position` function returns the position of the `Language` node being currently processed. For each `Language` element node, its position within the nodeset of `Language` elements is output to the `language/@position` attribute node.

name, local-name, namespace-uri

These functions are all used the same way and return, respectively, the name, local-name, and namespace URI of the input node. The screenshot below shows how these functions are used. Notice that no context node is specified.

The name function returns the name of the `Language` node and outputs it to the `language/@elementname` attribute. If the argument of any of these functions is a nodeset instead of a single node, the name (or local-name or namespace URI) of the first node in the nodeset is returned.



The `name` function returns the QName of the node; the `local-name` function returns the local-name part of the node's QName. For example, if a node's QName is `altova:MyNode`, then `MyNode` is the local name.

The namespace URI is the URI of the namespace to which the node belongs. For example, the `altova:` prefix can be declared to map to a namespace URI in this way: `xmlns:altova="http://www.altova.com/namespaces"`.

Note: Additional XPath 1.0 functions can be found in the Core function library.

7.5.21 xslt | xslt functions

The functions in the XSLT Functions library are XSLT 1.0 functions.

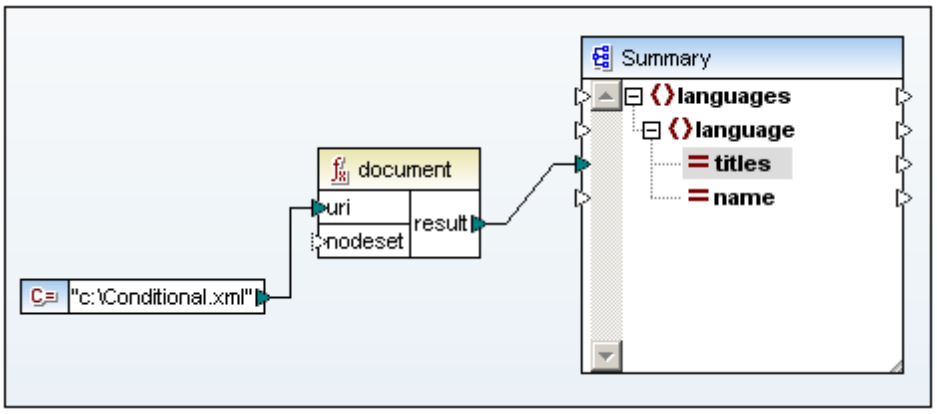
7.5.21.1 currrent

The **`current`** function takes no argument and returns the current node.

7.5.21.2 document

The **`document`** function addresses an external XML document (with the `uri` argument; *see screenshot below*). The optional nodeset argument specifies a node, the base URI of which is used to resolve the URI supplied as the first argument if this URI is relative. The result is output to a node in the output document.

 © 2015 Altova GmbH

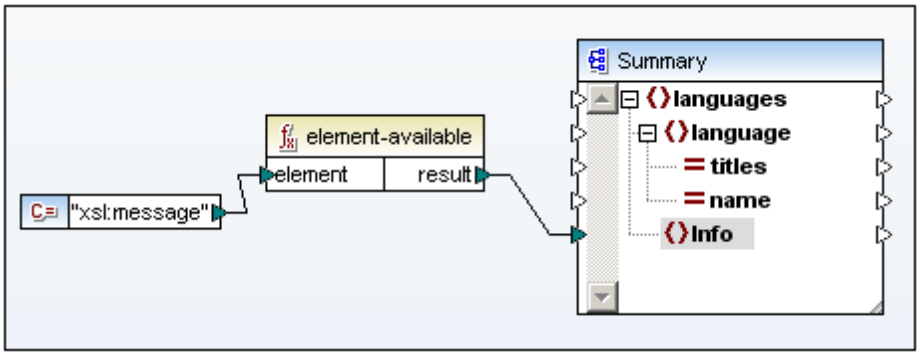


Note that the `uri` argument is a string that must be an absolute file path.

7.5.21.3 element-available

The **`element-available`** function tests whether an element, entered as the only string argument of the function, is supported by the XSLT processor.

The argument string is evaluated as a QName. Therefore, XSLT elements must have an `xsl:` prefix and XML Schema elements must have an `xs:` prefix—since these are the prefixes declared for these namespaces in the underlying XSLT that will be generated for the mapping.



The function returns a boolean.

7.5.21.4 function-available

The **`function-available`** function is similar to the **`element-available`** function and tests whether the function name supplied as the function's argument is supported by the XSLT processor.

The input string is evaluated as a QName. The function returns a boolean.

7.5.21.5 generate-id

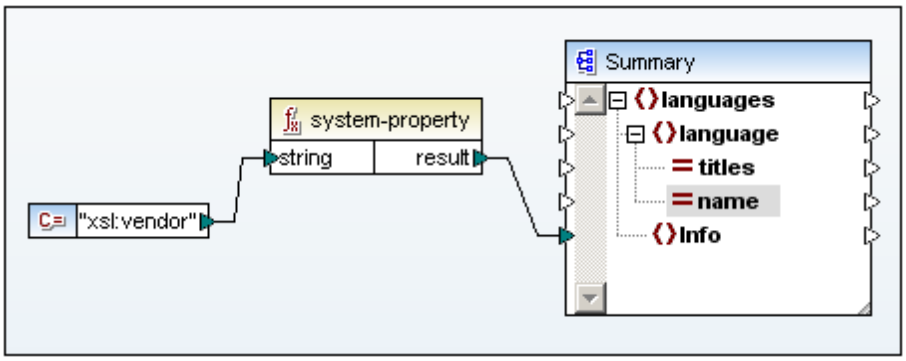
The **`generate-id`** function generates a unique string that identifies the first node in the nodeset identified by the optional input argument.

If no argument is supplied, the ID is generated on the context node. The result can be directed to any node in the output document.

7.5.21.6 system-property

The **system-property** function returns properties of the XSLT processor (the system). Three sytsem properties, all in the XSLT namespace, are mandatory for XSLT processors. These are `xsl:version`, `xsl:vendor`, and `xsl:vendor-url`.

The input string is evaluated as a QName and so must have the `xsl:prefix`, since this is the prefix associated with the XSLT namespace in the underlying XSLT stylesheet.



7.5.21.7 unparsed-enity-uri

If you are using a DTD, you can declare an unparsed entity in it. This unparsed entity (for example an image) will have a URI that locates the unparsed entity.

The input string of the function must match the name of the unparsed entity that has been declared in the DTD. The function then returns the URI of the unparsed entity, which can then be directed to a node in the output document, for example, to an `href` node.

 © 2015 Altova GmbH

Chapter 8

Automating Mappings and MapForce

8 Automating Mappings and MapForce

This section describes the command line interface of MapForce and ways to automate mapping execution with RaptorXML Server.

© 2015 Altova GmbH

8.1 Automation with RaptorXML Server

RaptorXML Server (hereafter also called RaptorXML for short) is Altova's third-generation, super-fast XML and XBRL processor. It has been built to be optimized for the latest standards and parallel computing environments. Designed to be highly cross-platform capable, the engine takes advantage of today's ubiquitous multi-core computers to deliver lightning fast processing of XML and XBRL data.

RaptorXML is available in several editions which can be downloaded and installed from the Altova download page:

- RaptorXML Server is a very fast XML processing engine with support for XML, XML Schema, XSLT, XPath, XQuery, and more. This edition is part of the FlowForce Server installation package.
- RaptorXML+XBRL Server supports all the features of RaptorXML Server with the additional capability of processing and validating the XBRL family of standards.

Limitations:

- XML Signatures are not supported
- Global resources are not supported via the COM interface
- ODBC and ADO database connections are only supported on Windows. On other operating systems, JDBC must be used.

If you generate code in XSLT 1.0 or 2.0, MapForce creates a batch file called `DoTransform.bat` which is placed in the output folder that you choose upon generation. Executing the batch file calls RaptorXML Server and executes the XSLT/XQuery transformation on the server.

If you intend to execute or automate MapForce mappings for other outputs on a server, refer to Altova MapForce Server and FlowForce Server.

Note: You can also preview the XSLT code using the built-in engine.

8.2 MapForce Command Line Interface

General command line syntax:

MapForce.exe *filename* [*/target* [*outputdir*] *options*

- Square brackets **[...]** denote optional parameters.
- Curly brackets **{...}** denote a parameter group containing several choices.
- The **pipe** symbol **|** denotes OR, e.g. /XSLT or /JAVA

MapForce.exe returns an exit code of 0, if the command line execution was successful. Any other value indicates a failure. You can check for this code using the IF ERRORLEVEL command in batch files.

filename
The MFD or MFP file to load. **If the path, or file name contains a space, please use quotes around the path/file name i.e. "c:\Program Files\...\Filename"**

target

/XSLT	generates XSLT 1.0 code
/XSLT2	generates XSLT 2.0 code

/**GLOBALRESOURCEFILE** *globalresourcefilename*	uses the global resources defined in the specified global resource file
/**GLOBALRESOURCECONFIG** *configurationname*	uses the specified global resource configuration

outputdir
The directory the generated mapping code is to be placed in outputdir is optional. If an output path is not supplied, the working/current directory will be used. If not specified, relative file names are relative to the working/current directory. Relative means only the file name is supplied, not a complete path starting with a drive letter).

options
Specifies various options:

/**LOG** *logfilename*	Generates a log file called *logfilename. Logfilename* can be a full path name, i.e. directory and file name of the log file, but the directory must exist for the logfile to be generated if a full path is supplied. If you only specify the file name, then the file will be placed in the */outputdir* directory.

Examples:

MapForce.exe *filename* starts MapForce and opens the file defined by *filename*.

 © 2015 Altova GmbH

I) generate all XSLT files and output a log file.

MapForce.exe *filename* **/XSLT** *outputdir* **/LOG** *logfilename*

II) generate all XSLT files and use all global resources of the global resource file for the specified configuration

Mapforce.exe *filename* **/XSLT** *outputdir* **/GLOBALRESOURCEFILE** *globalresourcefilename* **/GLOBALRESOURCECONFIG** *configurationname*

Having executed the command line, several files are produced:

- The generated XSLT file
- A batch file called **DoTransform.bat** which uses RaptorXML Server to transform the XML file.
- A log file, if one was specified in the command line.

To transform the XML file using the generated XSLT:

1. Download and install the RaptorXML Server engine from the RaptorXML download page.
2. Start the DoTransform.bat batch file located in the previously designated output folder. This generates the output file in the current folder.

Note:

You might need to add the RaptorXML installation location to the path variable of the Environment Variables.

Chapter 9

Customizing MapForce

9 Customizing MapForce

This section provides information about working with Altova Global Resources, and working with catalog files.

© 2015 Altova GmbH

9.1 Altova Global Resources

Global resources are a major enhancement in the interoperability between products of the Altova product line, and are currently available in the following Altova products: XMLSpy, MapForce, StyleVision and DatabaseSpy. Users of the Altova MissionKits have access to the same functionality in the respective products.

General uses:

- Workflows can be defined that use various Altova applications to process data.
- An application can invoke a target application, initiate data processing there, and route the data back to the originating application.
- Defining input and output data, file locations , as global resources.
- Switching between global resources during runtime to switch between development or deployment resources.
- What-if scenarios for QA purposes.

The default location of the global resource definitions file, **GlobalResources.xml**, is c:\Documents and Settings\UserName\My Documents\Altova\. This is the default location for all Altova applications that can use global resources. Changes made to global resources are thus automatically available in all applications. The file name and location can be changed. Please see Global Resources - Properties for more information.

General mechanism:

- Global resources are **defined** in an application and automatically saved.
- Global resources are **assigned** to components whose data you intend to be variable.
- The global resource is invoked / **activated** in an application, allowing you to switch resources at runtime.

This section will describe how to define and use, global resources using existing mappings available in the ...\MapForceExamples\Tutorial\ folder.

To activate the Global Resources toolbar:
Select the menu option **Tools | Customize |** click the **Toolbar tab** and activate the Global Resources check box. This displays the global resources tool bar.

The combo box allows you to switch between the various resources, a "Default" entry is always available.
Clicking the Global Resources icon opens the Global Resources dialog box (alternatively Tools | Global Resources).

9.1.1 Global Resources - Files

Global Resources in MapForce:

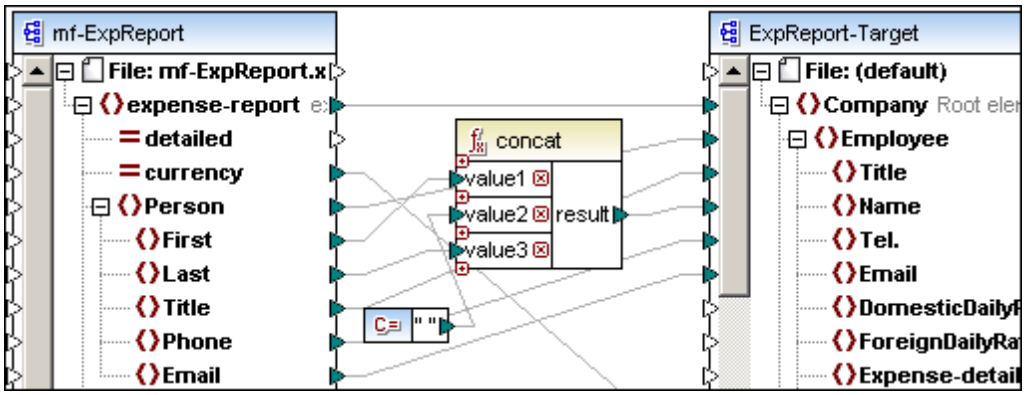
- Any input/output components **files** can be defined as global resources, e.g. XML, XML Schema, files.

Aim of this section:

- To make the source component input file, **mf-ExpReport**, a global resource.

- To **switch** between the two XML files that supply its **input data** at runtime, and check the resulting XML output in the Output preview tab.

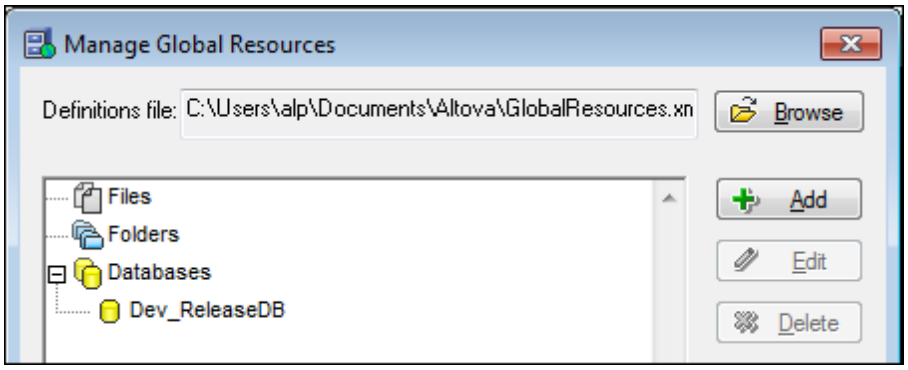
  This section uses the **Tut-ExpReport.mfd** file available in the ...\MapForceExamples\Tutorial\ folder.



9.1.1.1 Defining / Adding global resources

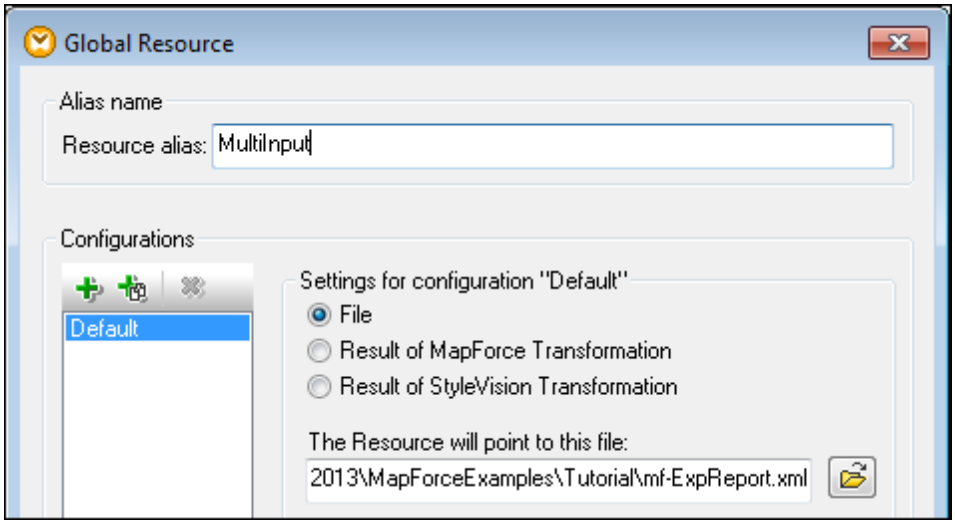
Defining / Adding a global resource file:

1. Click the Global Resource icon to open the dialog box.

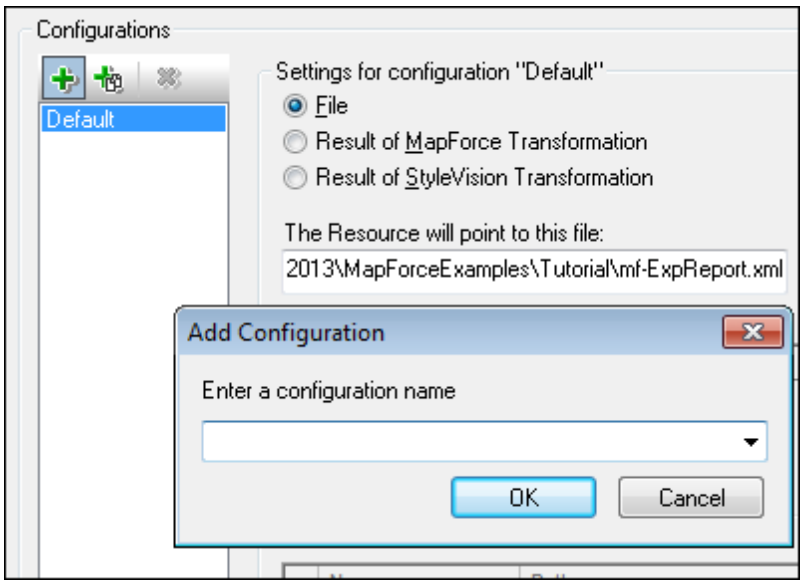


   This is the state of an empty global resources file.
2. Click the **Add** button and select **File** from the popup.
3. Enter the name of the Resource alias e.g. **MultiInput**.
4. Click the Open folder icon and select the XML file that is to act as the "Default" input file e.g. **mf-ExpReport.xml**.

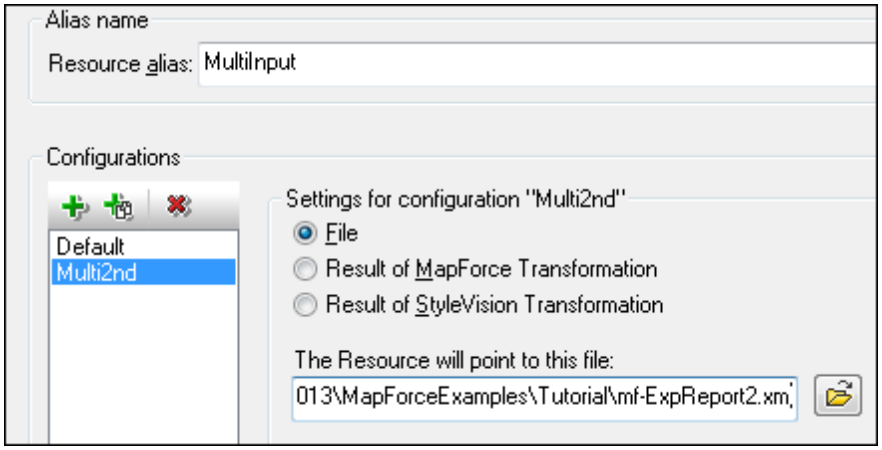
© 2015 Altova GmbH



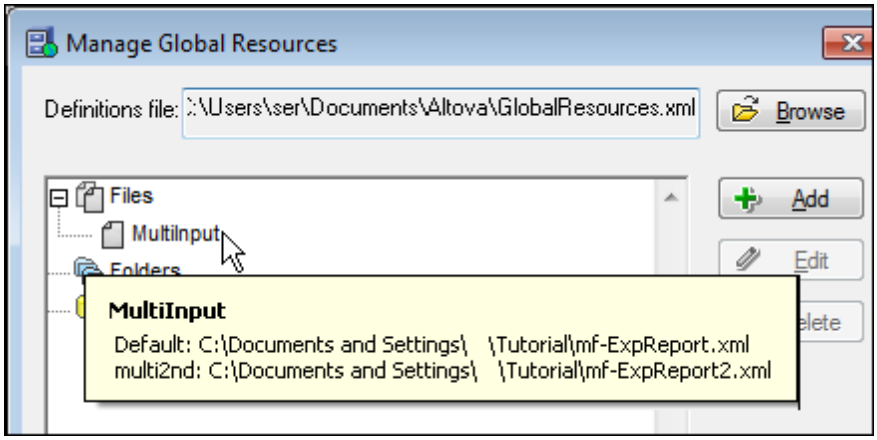
5. Click the Add button of the **Configurations** group, to add a new configuration to the current Alias. Note that the Copy configuration icon, allows you to copy a selected configuration and save it under a new name.



6. Enter a name for the configuration, **Multi2nd**, and click OK to confirm.
   Multi2nd has now been added to the Configurations list.
7. Click the Open folder icon again and select the XML file that is to act as the input file for the multi2nd configuration e.g. **mf-ExpReport2.xml**.



8. Click OK to complete the definition of the resource.
   The **MultiInput** alias has now been added to the Files section of the global resources. Placing the mouse cursor over an alias entry, opens a tooltip showing its defintion.



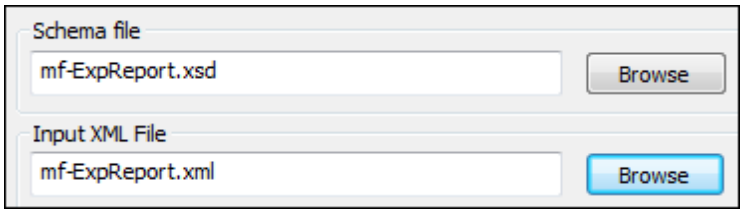
9. Click OK to confirm.
   This concludes the definition part of defining global resources. The next step is Assigning a global resource to a component.

9.1.1.2 Assigning a global resource

Assigning global resources to a component

We now have to assign the global resource to the component that is to make use of it, i.e. the mf-ExpReport.xml file that is being used as a source file for the mapping.

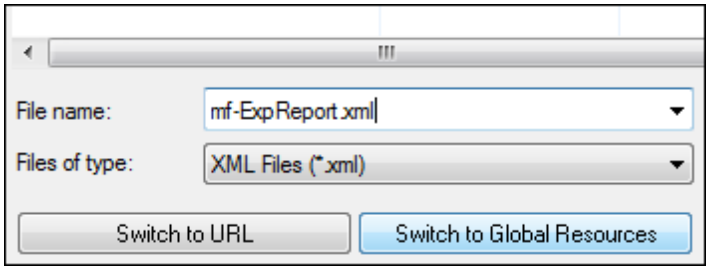
1. Double click the **mf-ExpReport** component and click the Browse button next to the Input XML File field.



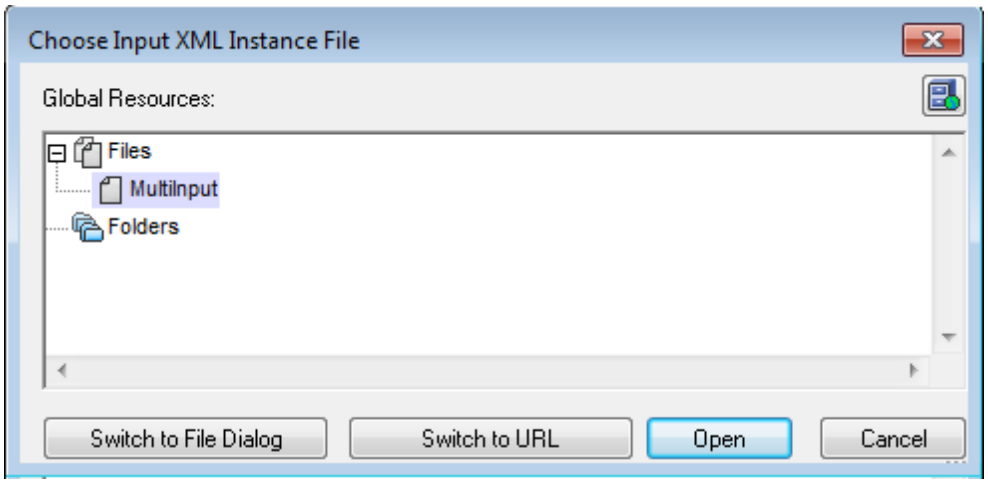
 © 2015 Altova GmbH

This opens the "Choose XML Instance file" dialog box.

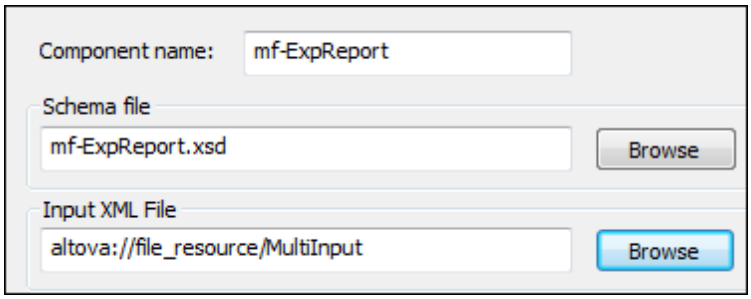
2. Click the **Switch to Global Resources** button at the base of the dialog box.



3. Click the resource you want to assign, **MultiInput** in this case, and click Open.



Note: the **Input XML File** field of the component, now contains a reference to a resource i.e. **altova://file_resource/MultiInput**.



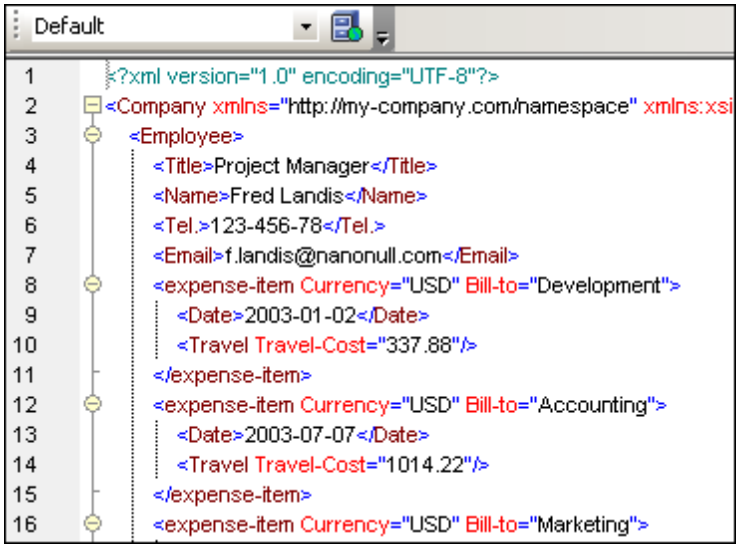
4. Click OK to complete the assignment of a resource to a component.
The next step is Using / activating a global resource.

9.1.1.3 Using / activating a global resource

Using / activating a global resource

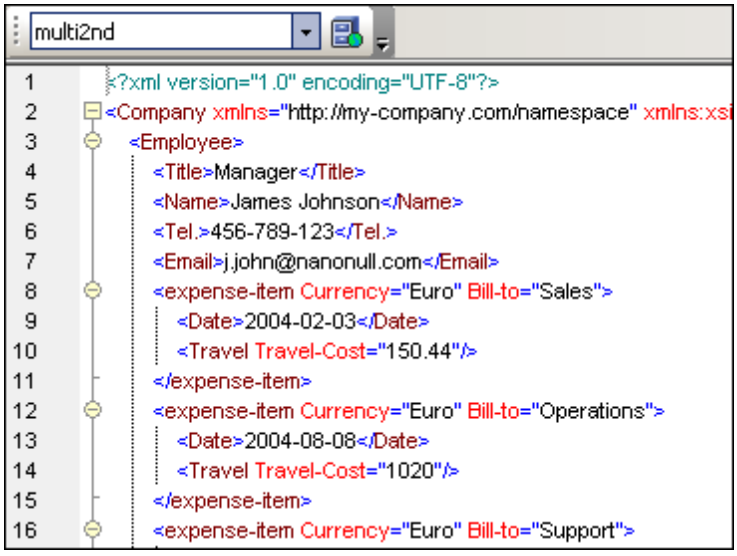
At this point the previously defined **Default** configuration for the **MultiInput** Alias is active. You can check this by noting that the entry in the Global Resources icon bar is "Default".

1. Click the Output tab to see the result of the mapping.



2. Click the **Mapping** tab to return to the mapping view.
3. Click the global resources combo box select **multi2nd** from the combo box.

4. Click the Output tab to see the new result.
   The mf-ExpReport2.xml file is now used as the source component for the mapping, and produces different output.



Note:

The **currently active** global resource (**multi2nd** in the global resources toolbar) determines the result of the mapping. This is also the case when you generate code.

© 2015 Altova GmbH

9.1.2 Global Resources - Folders

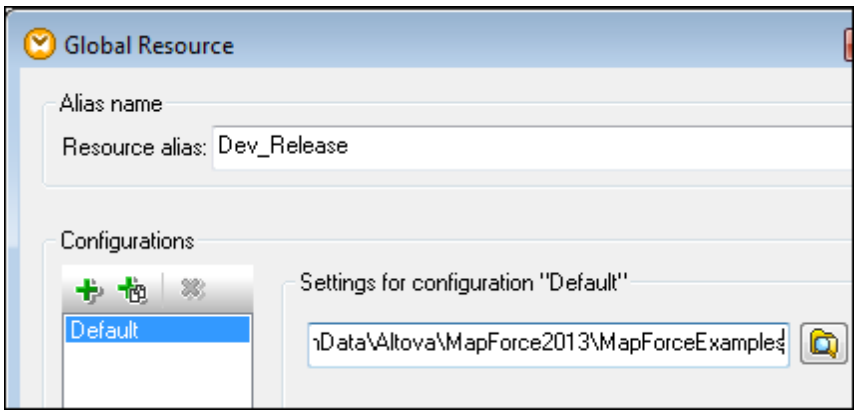
Folders can also be defined as a global resource, which means that input components can contain files that refer to different folders, for development and release cycles for example.

Defining folders for output components is not really useful in MapForce, as you are always prompted for the target folders when generating XSLT or code for other programming languages.

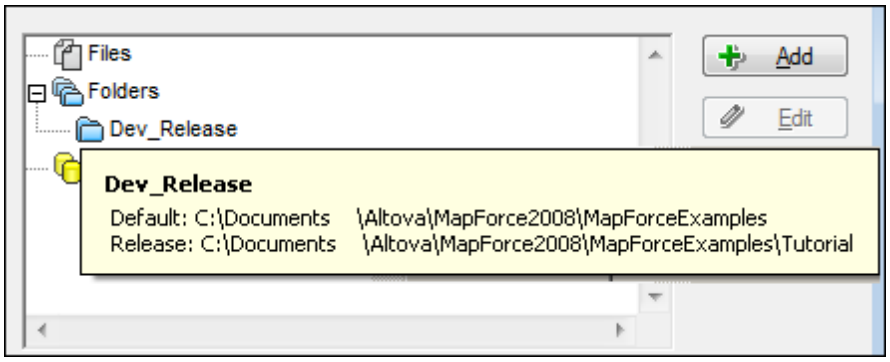
The mapping file used in this section is available as "**global-folder.mfd**" in the ...\MapForceExamples\Tutorial\ folder.

Defining / Adding global resource folders

1. Click the Global Resource icon to open the dialog box.
2. Click the **Add** button and select **Folder** from the popup.
3. Enter the name of the Resource alias e.g. **Dev_Release**.
4. Click the Open folder icon and select the "Default" input folder, ...\MapForceExamples.



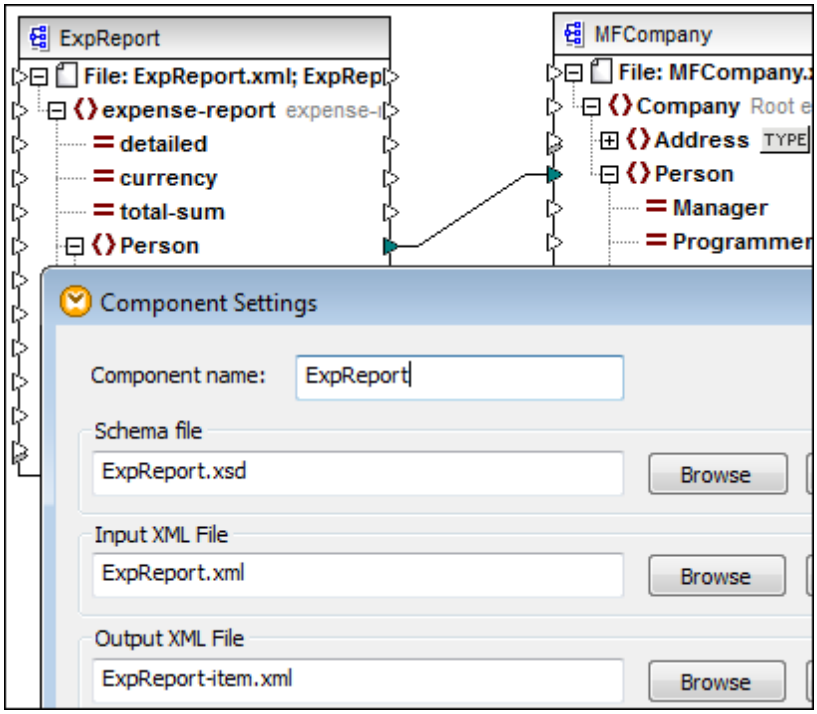
5. Click the Add button of the **Configurations** group, to add a new configuration to the current Alias, and enter a name for it e.g. Release. Note that the Copy configuration icon , allows you to copy a selected configuration and save it under a new name.
6. Click the Open folder icon and select the Release input folder, ...\MapForceExamples\Tutorial.



7. Click OK to finish the global folder definition.

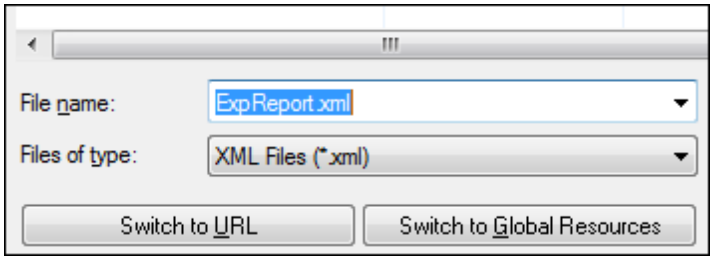
Assigning the global resource folders:

1. Double click the **ExpReport** component and click the **Browse** button next to the **Input XML File** field.

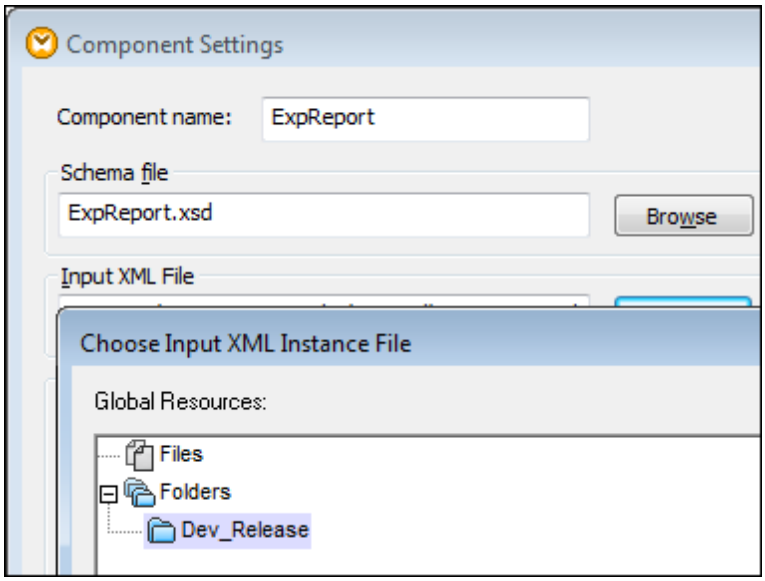


This opens the "Choose XML Instance file" dialog box.

2. Click the **Switch to Global Resources** button at the base of the dialog box.



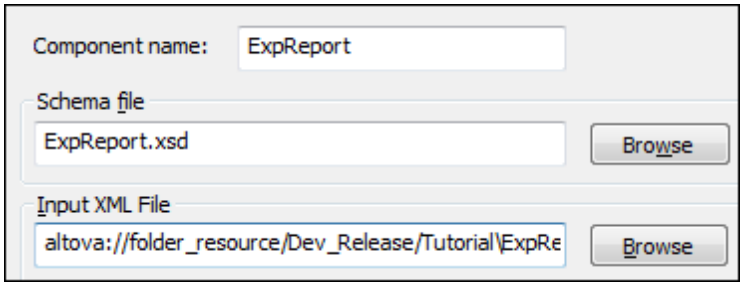
3. Click the resource you want to assign, **Dev_Release** in this case, and click OK.



 © 2015 Altova GmbH

The "Open..." dialog appears.

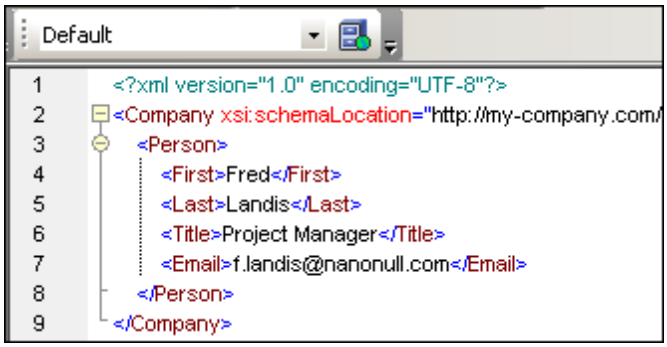
4. Select the **file** name that is to act as both the Development and Release **resource** file in each of the folders, e.g. **ExpReport.xml** and click OK to finish assigning the resource folder.



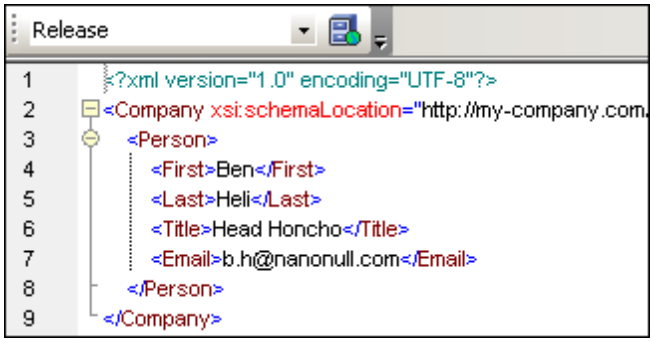
Note that this file is available in both folders but has different content.

Changing the resource folder at runtime:

1. Click the Output tab to see the result of the transformation.
   Note that this is the **Default** configuration/folder .../MapforceExamples.



2. Click the Mapping tab to return to the mapping window.
3. Click the Global Resource combo box and select the "**Release**" entry.
3. Click the Output button to see the result using the Release global resource.

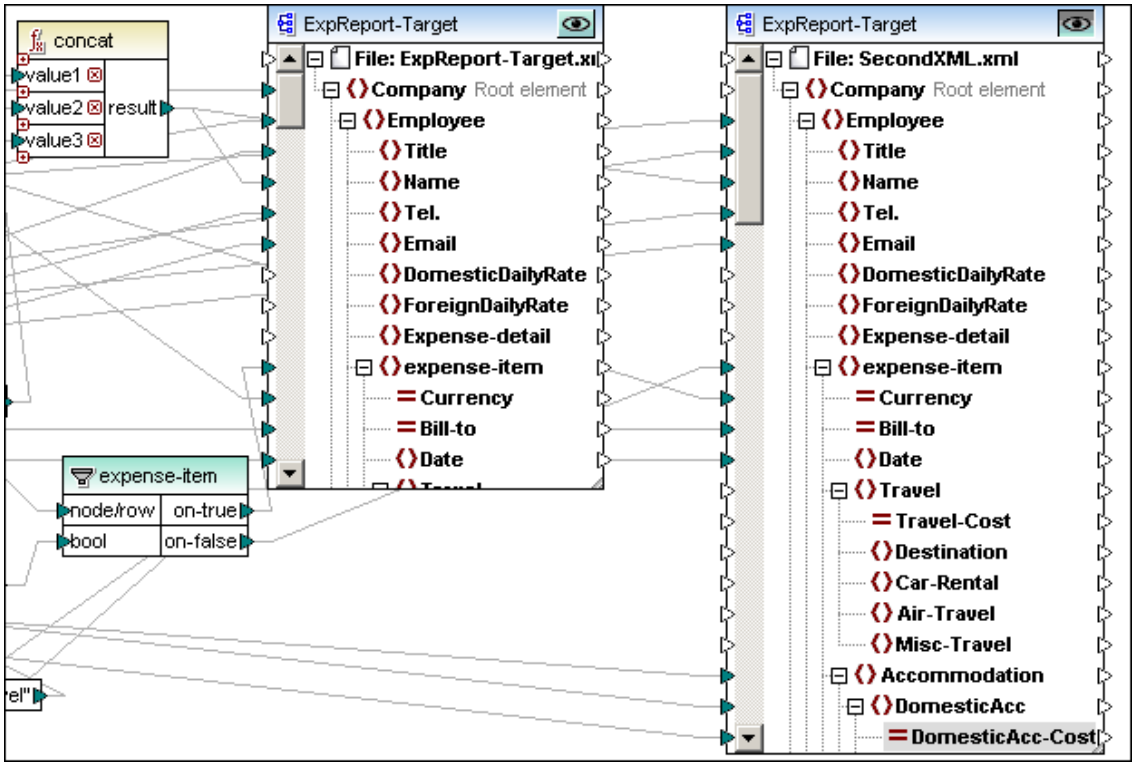


The output from the "Release" folder .../MapforceExamples/Tutorial is now displayed.

9.1.3 Global Resources - Application workflow

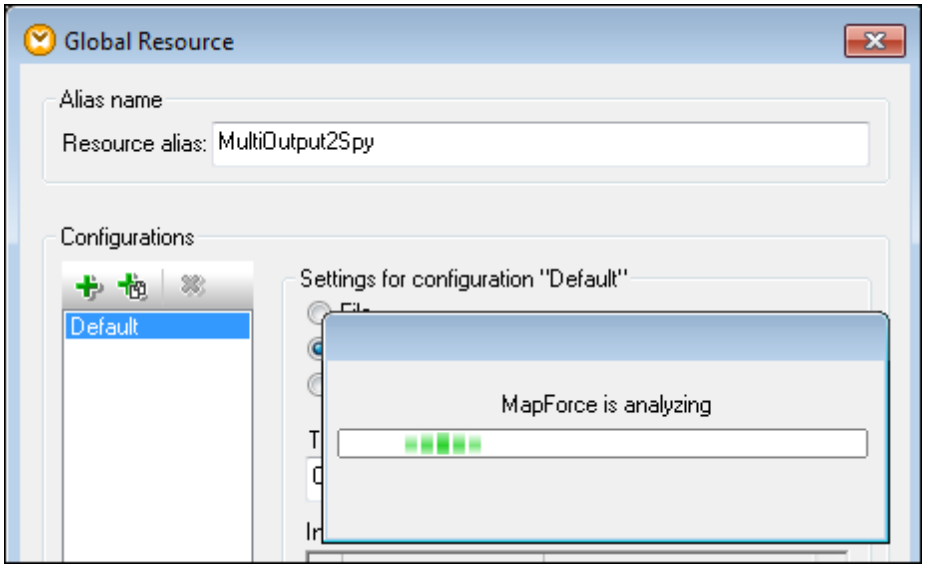
The aim of this section is to create a workflow situation between two Altova applications. Workflow is initiated in XMLSpy which starts MapForce and passes the generated XML file output back to XMLSpy for further processing.

This mapping uses two output components to produce two types of filtered output; Travel and Non-travel expenses of the expense report input file. This section uses the **Tut-ExpReport-multi.mfd** mapping file available in the ...\MapForceExamples\Tutorial\ folder.

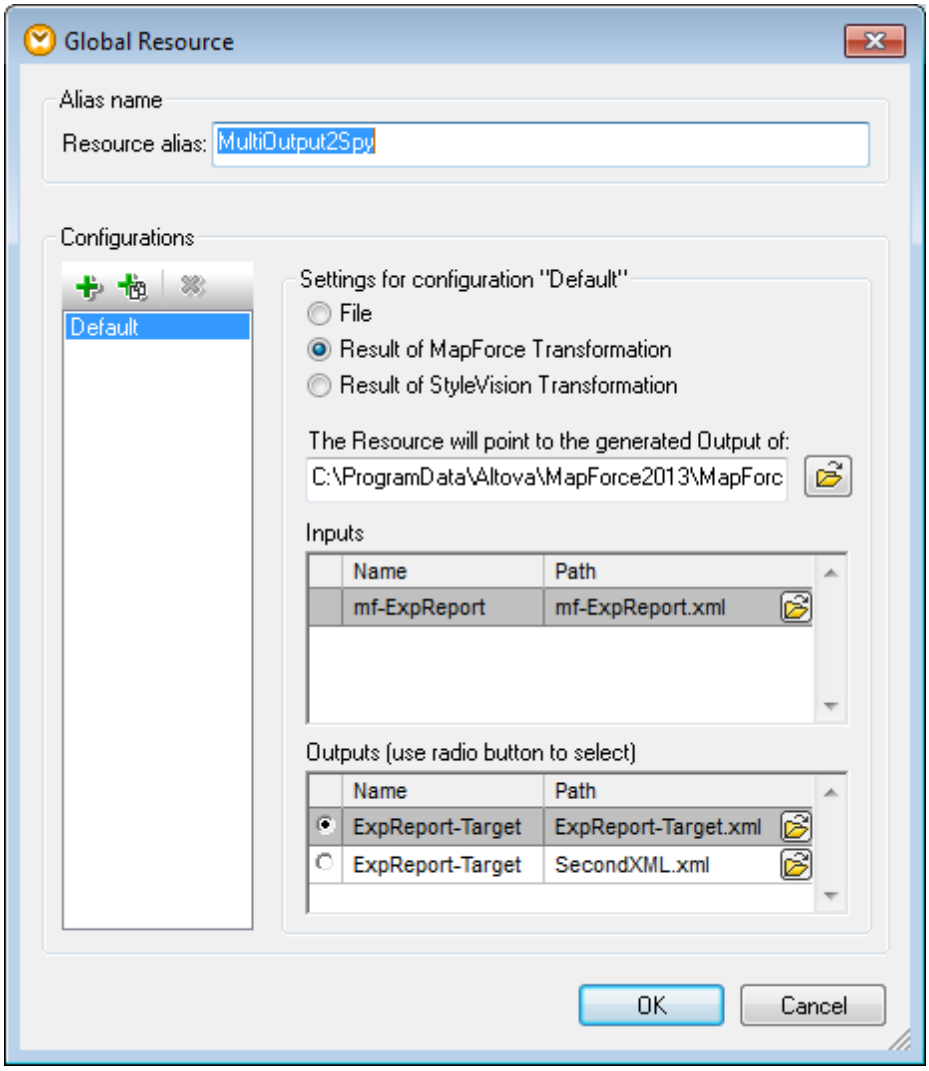


1. Click the Global Resource icon to open the dialog box.
2. Click the **Add** button and select **File** from the popup.
3. Enter the name of the Resource alias e.g. **MultiOutput2Spy**
4. Click the "**Result of MapForce Transformation**" radio button, then click the Open file icon.
5. Select the **Tut-ExpReport-multi.mfd** mapping.

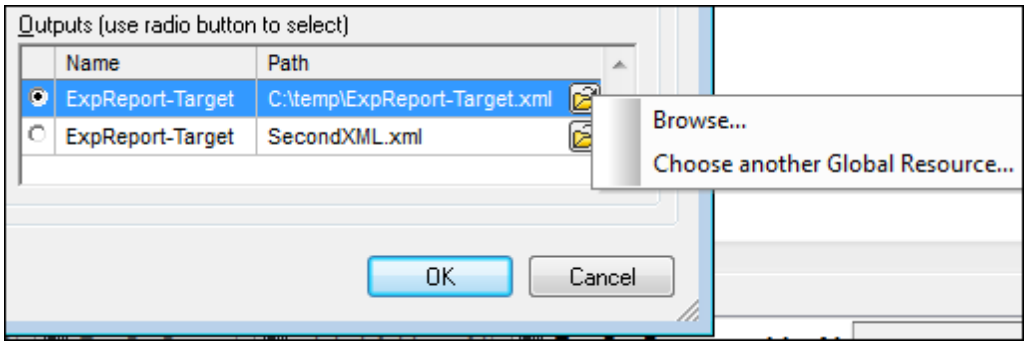
 © 2015 Altova GmbH



MapForce analyzes the mapping and displays the input and output files in separate list boxes.



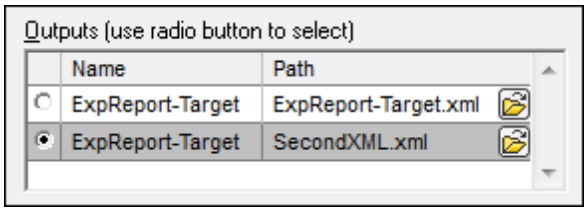
6. Click the top radio button entry in the **Outputs** section, if not already selected. Note that the output file name is **ExpReport-Target.xml** and that we are currently defining the **Default** configuration.
7. Click the icon and select Browse from the popup menu, to define the new location of the output file e.g. C:\Temp.



8. Enter the new output location e.g. C:\Temp and click the Save button. This location can differ from the location defined in the component settings.

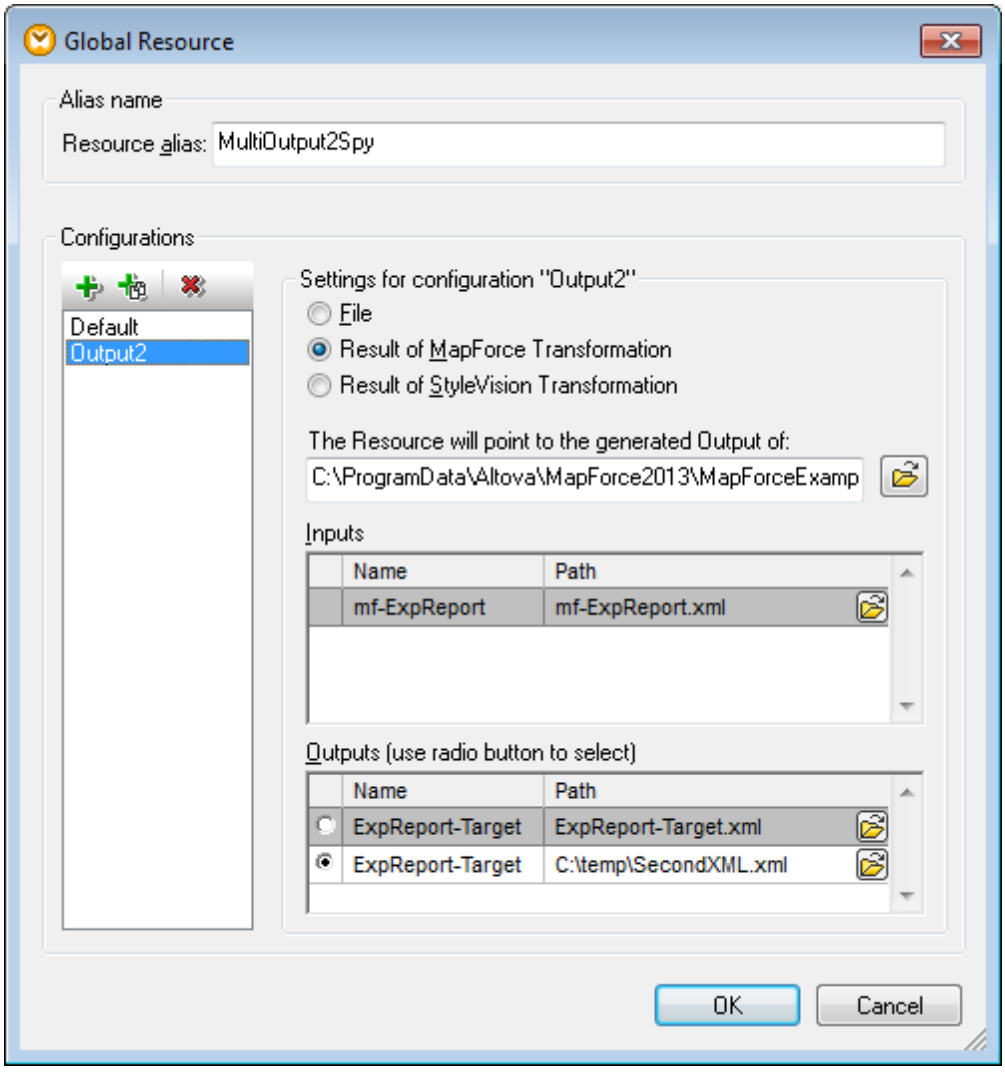
 © 2015 Altova GmbH

9. Click the Add button of the Configurations group (of this dialog box), to add a new configuration to the resource alias.
10. Enter the name of the configuration, e.g. **Output2**, click the Open file icon, select the Tut-ExpReport-multi.mfd.
11. Click the lower radio button of the Outputs listbox. Note that the output file name is **SecondXML.xml**.

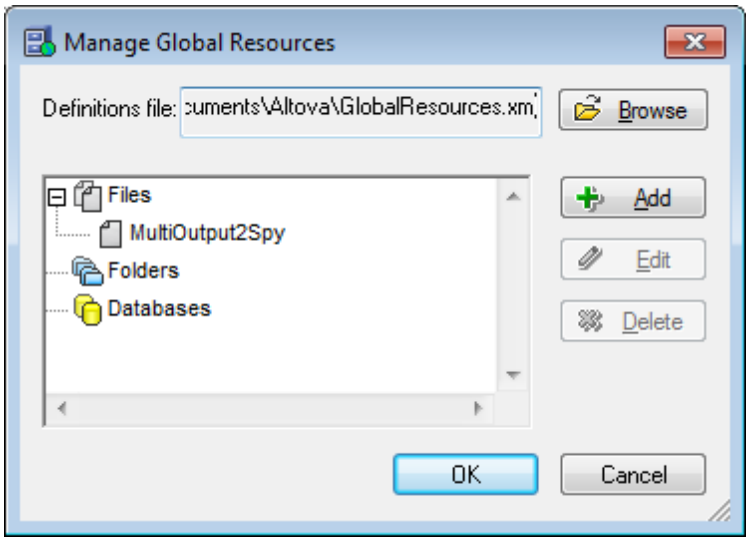


12. Click the icon and select Browse from the popup menu, to define the new location of the output file e.g. C:\Temp.

    Note: clicking the "**Choose another Global Resource...** in the popup, allows you to save the MapForce output as a global resource. I.e. the output is stored to a file that the global resource physically points to/references.



13. Click OK to save the new global resources.
    The new resource alias **MultiOutput2Spy** has been added to the Global Resources definition file.



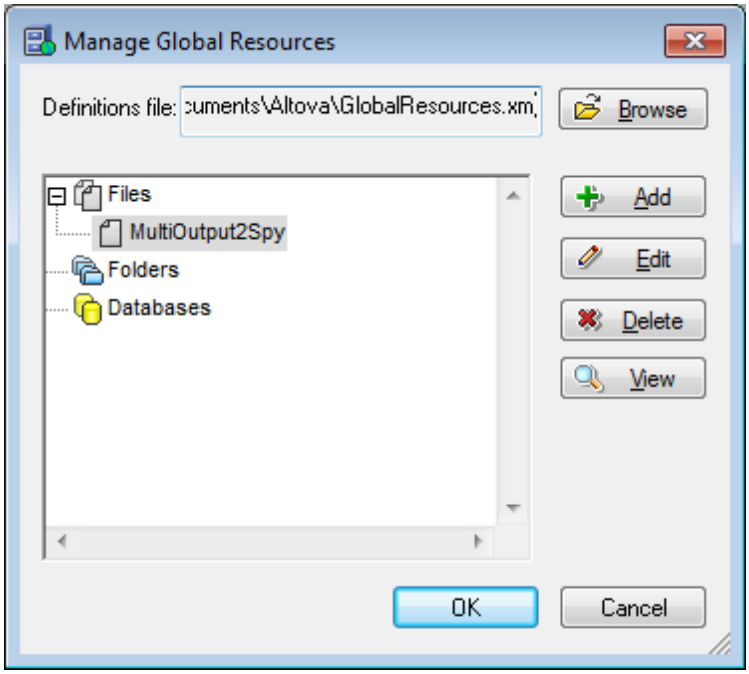
14. Click OK to complete the definition phase.

9.1.3.1 Start application workflow

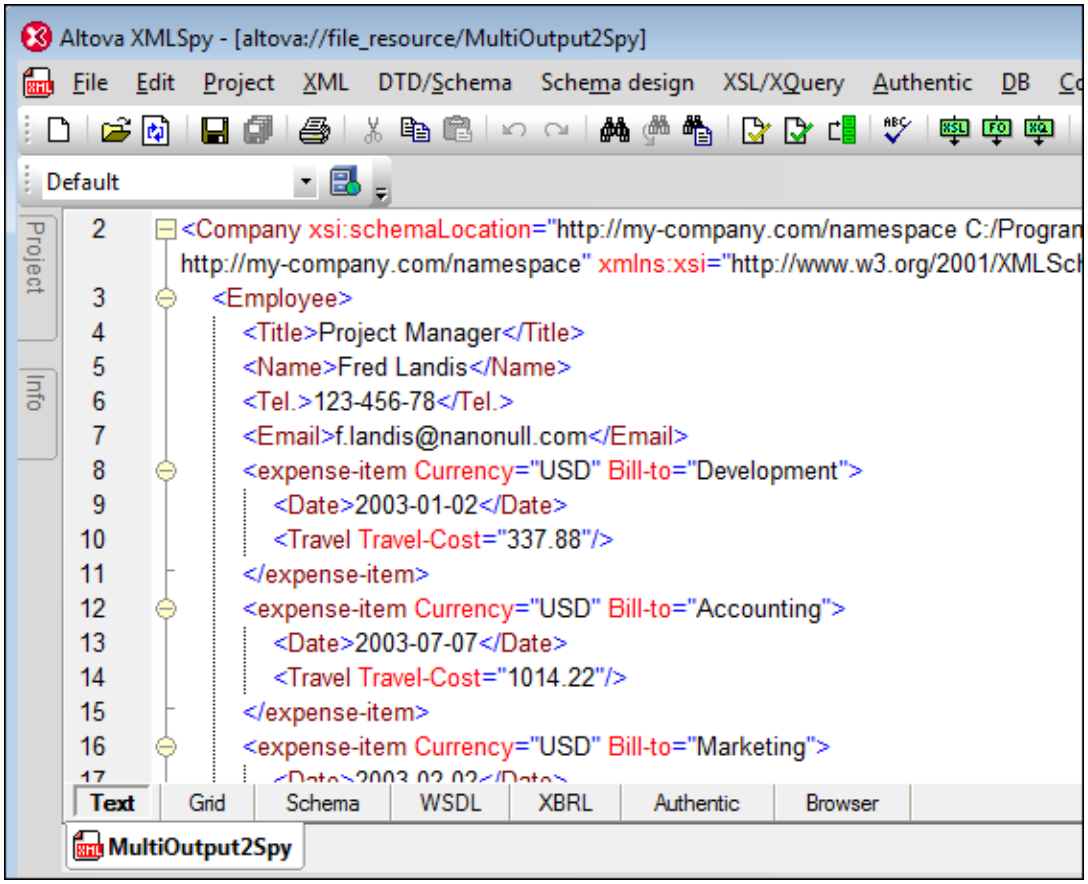
This section shows how the Global Resource is activated in XMLSpy and how the resulting MapForce transformation is routed back to it.

1. Start XMLSpy and shut down MapForce, if open, to get a better view of how the two applications interact.
2. Select the menu option **Tools | Global Resources** in XMLSpy.
3. Select the **MultiOutput2Spy** entry, and click the **View** button.

© 2015 Altova GmbH



A message box stating that MapForce is transforming appears, with the result of the transformation appearing in the Text view window.

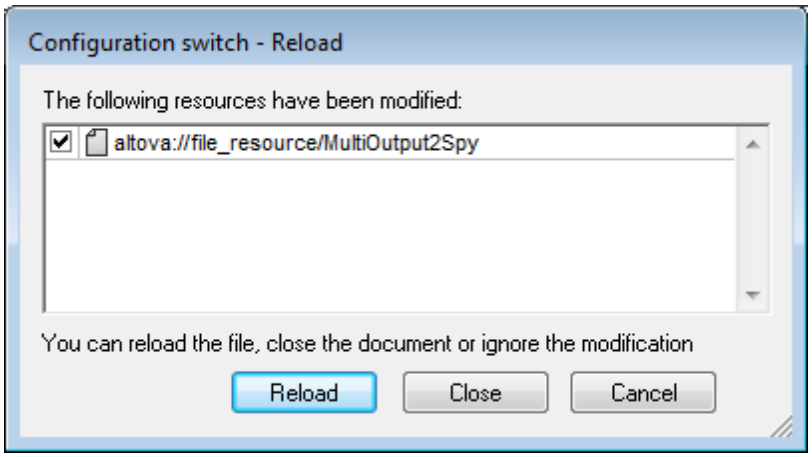


Note:

- The currently selected configuration is "**Default**".
- The name of the resource alias is in the application title bar **altova://file_resource/MultiOutput2Spy**.
- The output file has been opened as "MultiOutput2Spy.xml" for further processing.
- The **ExpReport-Target.xml** file has been copied to the C:\Temp folder.

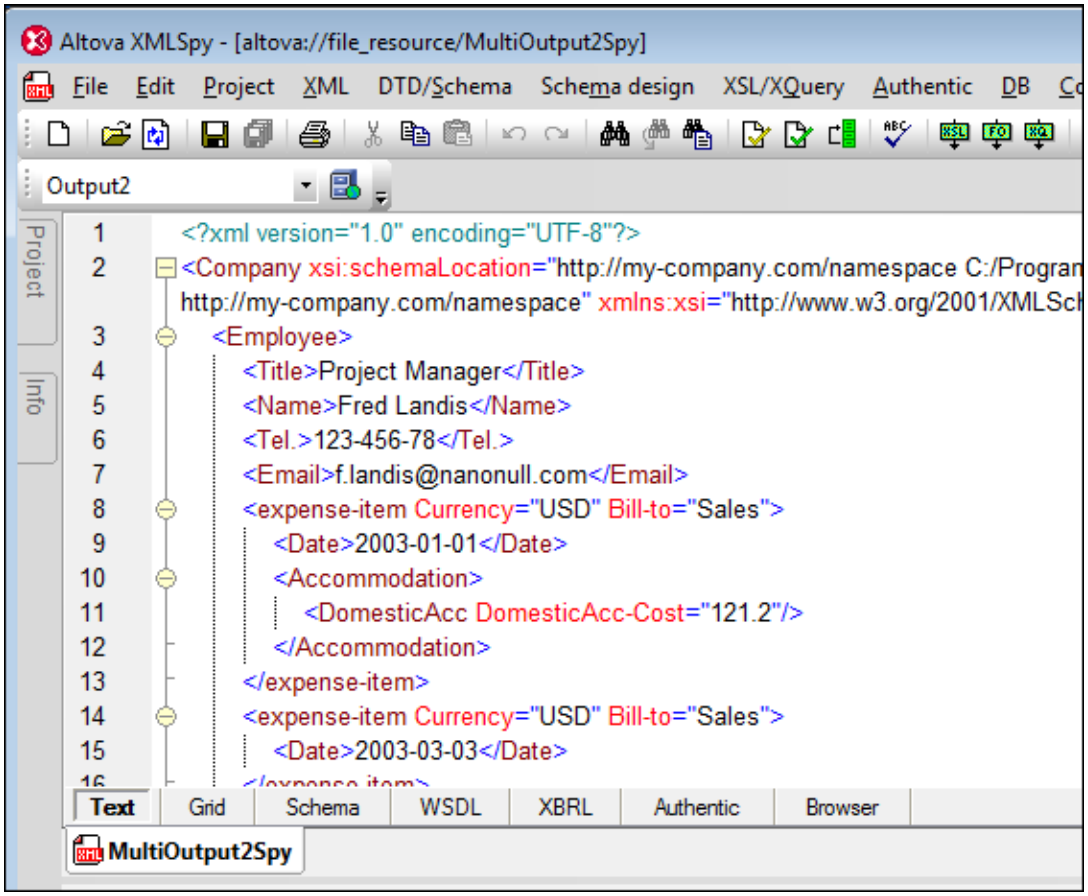
To retrieve the non-travel expenses output:

1. Click the Global Resources combo box and select "**Output2**".
   A notification message box opens.



2. Click **Reload** to retrieve the second output file defined by the resource.

 © 2015 Altova GmbH



The result of the transformation appears in the Text view window and overwrites the previous MultiOutput2Spy.xml file.

Note:

- The currently selected configuration is "**Output2**"
- The output file has been opened as "Untitled1.xml" for further processing.
- The **SecondXML.xml** file has been copied to the C:\Temp folder.

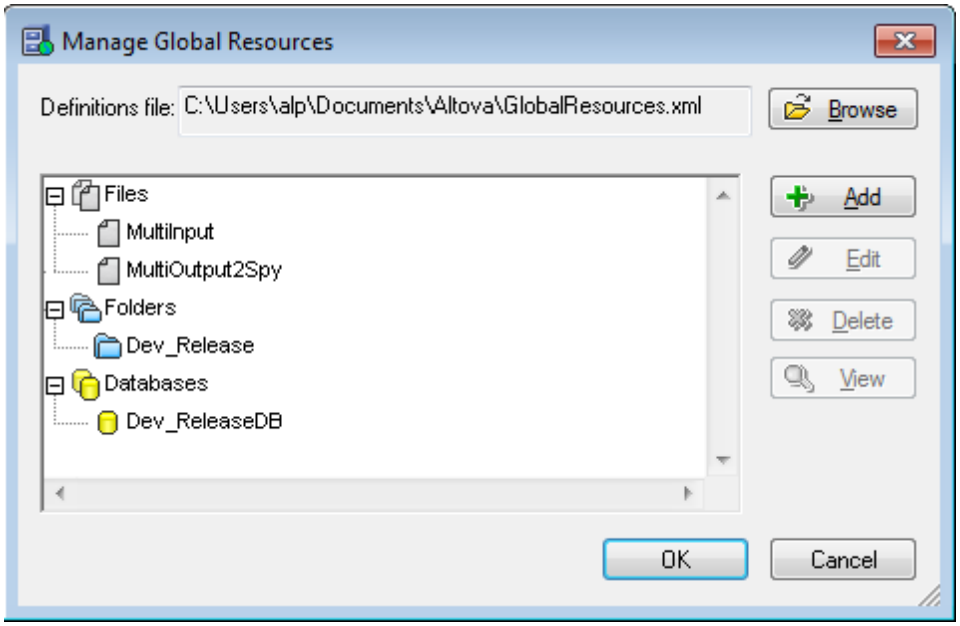
9.1.4 Global Resources - Properties

The Global Resources XML File

Global resources definitions are stored in an XML file. By default, this XML file is called `GlobalResources.xml`, and it is stored in the folder `C:\Documents and Settings\<username>\My Documents\Altova\`. This file is set as the default Global Resources XML File for all Altova applications. As a result, a global resource defined in any application will be available to all Altova applications—assuming that all applications use this file.

You can also re-name the file and save it to any location. You can therefore have multiple Global Resources XML files. However, only one of these Global Resources XML File can be active, per application, at one time, and only the definitions contained in this file will be available to the application.

To make the Global Resources XML file active, click the **Browse** button of the "Definitions file" field and select the one you want to use from the "Open..." dialog box.



Managing global resources: adding, editing, deleting

In the Global Resources dialog, you can add a global resource to the selected Global Resources XML File, or edit or delete a selected global resource. The Global Resources XML File organizes the aliases you add into the following sections: files, folders.

To add a global resource:

Click the **Add** button and define the global resource in the Global Resource dialog that pops up. After you define a global resource and save it, the global resource (or alias) is added to the list of global definitions in the selected Global Resources XML File.

To edit a global resource:

Select it and click **Edit**. This pops up the Global Resource dialog, in which you can make the necessary changes.

To delete a global resource:

Select it and click **Delete**.

To view the result of an application workflow:

If the calling application e.g. XMLSpy, calls another application e.g. MapForce, then a View button is available in the Manage Global Resources dialog box.

Clicking the View button shows the affect of the currently selected global resource in the callling application. Please see Global Resources - Application workflow for an example.

To save modifications made in the Managing Global Resources dialog box:

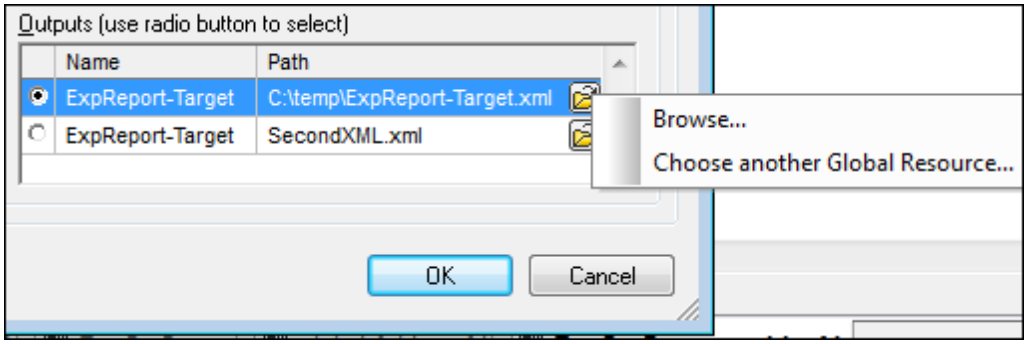
Having finished adding, editing, or deleting, make sure to click **OK** in the Global Resources dialog to save your modifications to the Global Resources XML File.

Note: Alias resource names must be unique **within** each of the Files, Folders. You can however define an identical alias name in two different sections, e.g. a multiInput alias can exist in the Files section as well as in the Folders section.

Selecting Results of MapForce transformations as a global resource

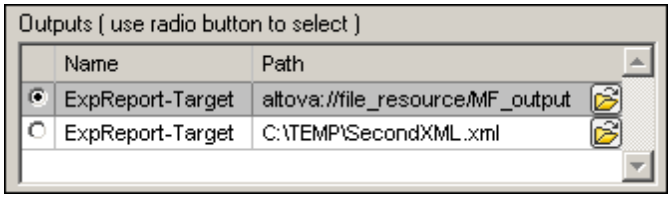
 © 2015 Altova GmbH

In a MapForce transformation that has multiple outputs, you can select which one of the output files should be used for the global resource by clicking its radio button.



The **output** file that is generated by the mapping can be saved as:

- a global resource via the **Choose another Global Resource** entry in the popup, visible as **altova://file_resource/MF_output**. The output is stored to a file that the global resource physically points to/references.



- a file via the icon, shown as C:\TEMP\Second.xml.

  If neither of these options is selected, a **temporary** XML file is created when the global resource is used.

Determining which resource is used at runtime

There are two application-wide selections that determine what global resources can be used and which global resources are actually used at any given time:

- *The active Global Resources XML File* is selected in the Global Resource dialog. The active Global Resources XML File can be changed at any time, and the global-resource definitions in the new active file will immediately replace those of the previously active file.

  The active Global Resources XML File therefore determines: (i) what global resources can be assigned, and (ii) what global resources are available for look-up (for example, if a global resource in one Global Resource XML File is assigned but there is no global resource of that name in the currently active Global Resources XML File, then the assigned global resource (alias) cannot be looked up).

- *The active configuration* is selected via the menu item **Tools | Active Configuration** or via the Global Resources toolbar. Clicking this command (or dropdown list in the toolbar) pops up a list of configurations across all aliases.

  Selecting a configuration makes that configuration active application-wide. This means that wherever a global resource (or alias) is used, the resource corresponding to the active configuration of each used alias will be loaded.

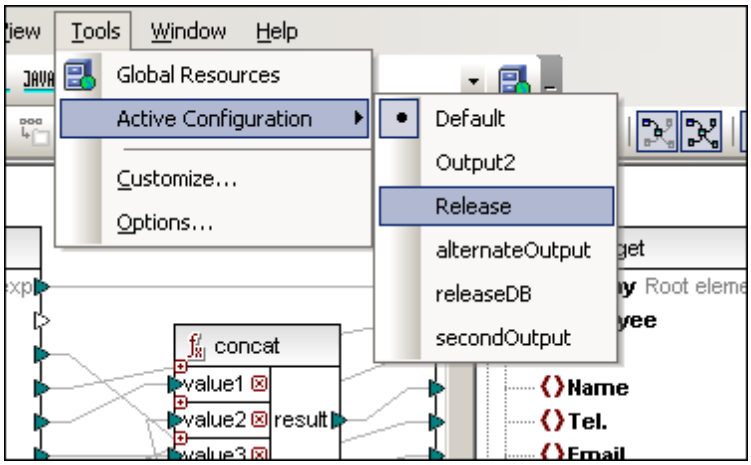
  The active configuration is applied to all used aliases. If an alias does not have a configuration with the name of the active configuration, then the **default configuration** of

that alias will be used. The active configuration is not relevant when assigning resources; it is significant only when the resources are actually used.

Changing resources / configurations

Resources can be switched by selecting a different configuration name. This can be done in two ways:

- When you hover over the menu command **Tools | Active Configuration**, a submenu with a list of all configurations in the Global Resources XML File appears. Select the required configuration from the submenu.



- In the combo box of the Global Resources toolbar, select the required configuration. The Global Resources toolbar can be toggled on and off with the menu command **Tools | Customize,** then click the **Toolbar tab** and enable/disable the Global resources check box.

 © 2015 Altova GmbH

9.2 Catalog Files

MapForce supports a subset of the OASIS XML catalogs mechanism. The catalog mechanism enables MapForce to retrieve commonly used schemas (as well as stylesheets and other files) from local user folders. This increases the overall processing speed, enables users to work offline (that is, not connected to a network), and improves the portability of documents (because URIs would then need to be changed only in the catalog files.)

The catalog mechanism in MapForce works as outlined below.

RootCatalog.xml

When MapForce starts, it loads a file called `RootCatalog.xml` (*structure shown in listing below*), which contains a list of catalog files that will be looked up. You can modify this file and enter as many catalog files to look up as you like, each in a `nextCatalog` element. Each of these catalog files is looked up and the URIs in them are resolved according to the mappings specified in them.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
     xmlns:spy="http://www.altova.com/catalog_ext"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="urn:oasis:names:tc:entity:xmlns:xml:catalog
Catalog.xsd">
  <nextCatalog catalog="%PersonalFolder%/Altova/%AppAndVersionName%/
CustomCatalog.xml"/>
  <nextCatalog catalog="CoreCatalog.xml"/>
  <!-- Include all catalogs under common schemas folder on the first directory
level -->
  <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/Schemas"
catalog="catalog.xml" spy:depth="1"/>
  <!-- Include all catalogs under common XBRL folder on the first directory
level -->
  <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/XBRL"
catalog="catalog.xml" spy:depth="1"/>
</catalog>
```

In the listing above, notice that in the `Schemas` and `XBRL` folders of the folder identified by the variable `%AltovaCommonFolder%` are catalog files named `catalog.xml`. (The value of the `%AltovaCommonFolder%` variable is given in the table below.)

The catalog files in the Altova Common Folder map the pre-defined public and system identifiers of commonly used schemas (such as SVG and WSDL) and XBRL taxonomies to URIs that point to locally saved copies of the respective schemas. These schemas are installed in the Altova Common Folder when MapForce is installed.You should take care not to duplicate mappings in these files, as this could lead to errors.

CoreCatalog.xml, CustomCatalog.xml, and Catalog.xml

In the `RootCatalog.xml` listing above, notice that `CoreCatalog.xml` and `CustomCatalog.xml` are listed for lookup:

- `CoreCatalog.xml` contains certain Altova-specific mappings for locating schemas in the

Altova Common Folder.

- `CustomCatalog.xml` is a skeleton file in which you can create your own mappings. You can add mappings to `CustomCatalog.xml` for any schema you require but that is not addressed by the catalog files in the Altova Common Folder. Do this using the supported elements of the OASIS catalog mechanism (*see below*).
- There are a number of `Catalog.xml` files in the Altova Common Folder. Each is inside the folder of a specific schema or XBRL taxonomy in the Altova Common Folder, and each maps public and/or system identifiers to URIs that point to locally saved copies of the respective schemas.

Location of catalog files and schemas

The files `RootCatalog.xml` and `CoreCatalog.xml` are installed in the MapForce application folder. The file `CustomCatalog.xml` is located in your `MyDocuments/Altova/MapForce` folder. The `catalog.xml` files are each in a specific schema folder, these schema folders being inside the folders: `%AltovaCommonFolder%\Schemas` and `%AltovaCommonFolder%\XBRL`.

Shell environment variables and Altova variables

Shell environment variables can be used in the `nextCatalog` element to specify the path to various system locations (*see RootCatalog.xml listing above*). The following shell environment variables are supported:

`%AltovaCommonFolder%`	C:\Program Files\Altova\CommonMapForce
`%DesktopFolder%`	Full path to the Desktop folder for the current user.
`%ProgramMenuFolder%`	Full path to the Program Menu folder for the current user.
`%StartMenuFolder%`	Full path to Start Menu folder for the current user.
`%StartUpFolder%`	Full path to Start Up folder for the current user.
`%TemplateFolder%`	Full path to the Template folder for the current user.
`%AdminToolsFolder%`	Full path to the file system directory that stores administrative tools for the current user.
`%AppDataFolder%`	Full path to the Application Data folder for the current user.
`%CommonAppDataFolder%`	Full path to the file directory containing application data for all users.
`%FavoritesFolder%`	Full path of the Favorites folder for the current user.
`%PersonalFolder%`	Full path to the Personal folder for the current user.
`%SendToFolder%`	Full path to the SendTo folder for the current user.

 © 2015 Altova GmbH

%FontsFolder%	Full path to the System Fonts folder.
%ProgramFilesFolder%	Full path to the Program Files folder for the current user.
%CommonFilesFolder%	Full path to the Common Files folder for the current user.
%WindowsFolder%	Full path to the Windows folder for the current user.
%SystemFolder%	Full path to the System folder for the current user.
%CommonAppDataFolder%	Full path to the file directory containing application data for all users.
%LocalAppDataFolder%	Full path to the file system directory that serves as the data repository for local (nonroaming) applications.
%MyPicturesFolder%	Full path to the MyPictures folder.

How catalogs work

Catalogs are commonly used to redirect a call to a DTD to a local URI. This is achieved by mapping, in the catalog file, public or system identifiers to the required local URI. So when the DOCTYPE declaration in an XML file is read, the public or system identifier locates the required local resource via the catalog file mapping.

For popular schemas, the PUBLIC identifier is usually pre-defined, thus requiring only that the URI in the catalog file point to the correct local copy. When the XML document is parsed, the PUBLIC identifier in it is read. If this identifier is found in a catalog file, the corresponding URL in the catalog file will be looked up and the schema will be read from this location. So, for example, if the following SVG file is opened in MapForce:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg width="20" height="20" xml:space="preserve">
  <g style="fill:red; stroke:#000000">
     <rect x="0" y="0" width="15" height="15"/>
     <rect x="5" y="5" width="15" height="15"/>
  </g>
</svg>
```

This document is read and the catalog is searched for the PUBLIC identifier. Let's say the catalog file contains the following entry:

```
<catalog>
  ...
    <public publicId="-//W3C//DTD SVG 1.1//EN" uri="schemas/svg/svg11.dtd"/>
  ...
</catalog>
```

In this case, there is a match for the PUBLIC identifier, so the lookup for the SVG DTD is redirected to the URI schemas/svg/svg11.dtd (this path is relative to the catalog file), and this local file will be used as the DTD. If there is no mapping for the Public ID in the catalog, then the URL in the XML document will be used (in the example above: http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd).

The catalog subset supported by MapForce

When creating entries in CustomCatalog.xml (or any other catalog file that is to be read by MapForce), use only the following elements of the OASIS catalog specification. Each of the elements below is listed with an explanation of their attribute values. For a more detailed explanation, see the XML Catalogs specification. Note that each element can take the xml:base attribute, which is used to specify the base URI of that element.

- <public publicId="PublicID of Resource" uri="URL of local file"/>
- <system systemId="SystemID of Resource" uri="URL of local file"/>
- <uri name="filename" uri="URL of file identified by filename"/>
- <rewriteURI uriStartString="StartString of URI to rewrite" rewritePrefix="String to replace StartString"/>
- <rewriteSystem systemIdStartString="StartString of SystemID" rewritePrefix="Replacement string to locate resource locally"/>

In cases where there is no public identifier, as with most stylesheets, the system identifier can be directly mapped to a URL via the system element. Also, a URI can be mapped to another URI using the uri element. The rewriteURI and rewritsSystem elements enable the rewriting of the starting part of a URI or system identifier, respectively. This allows the start of a filepath to be replaced and consequently enables the targeting of another directory. For more information on these elements, see the XML Catalogs specification.

File extensions and intelligent editing according to a schema

Via catalog files you can also specify that documents with a particular file extension should have MapForce's intelligent editing features applied in conformance with the rules in a schema you specify. For example, if you create a custom file extension .myhtml for (HTML) files that are to be valid according to the HTML DTD, then you can enable intelligent editing for files with these extensions by adding the following element of text to CustomCatalog.xml as a child of the <catalog> element.

```
<spy:fileExtHelper ext="myhtml" uri="schemas/xhtml/xhtml1-transitional.dtd"/>
```

This would enable intelligent editing (auto-completion, entry helpers, etc) of .myhtml files in MapForce according to the XHTML 1.0 Transitional DTD. Refer to the catalog.xml file in the %AltovaCommonFolder%\Schemas\xhtml folder, which contains similar entries.

XML Schema and catalogs

XML Schema information is built into MapForce and the validity of XML Schema documents is

 © 2015 Altova GmbH

checked against this internal information. In an XML Schema document, therefore, no references should be made to any schema for XML Schema.

The `catalog.xml` file in the `%AltovaCommonFolder%\Schemas\schema` folder contains references to DTDs that implement older XML Schema specifications. You should not validate your XML Schema documents against either of these schemas. The referenced files are included solely to provide MapForce with entry helper info for editing purposes should you wish to create documents according to these older recommendations.

More information

For more information on catalogs, see the XML Catalogs specification.

Chapter 10

Menu Reference

10 Menu Reference

This reference section contains a description of the MapForce menu commands.

© 2015 Altova GmbH

10.1 File

New
Creates a new mapping document.

Open
Opens previously saved mapping design (.mfd) files. Note that it is not possible to open mapping files which contain features not available in your MapForce edition.

Save
Saves the currently active mapping using the currently active file name.

Save As
Saves the currently active mapping with a different name, or allows you to supply a new name if this is the first time you save it.

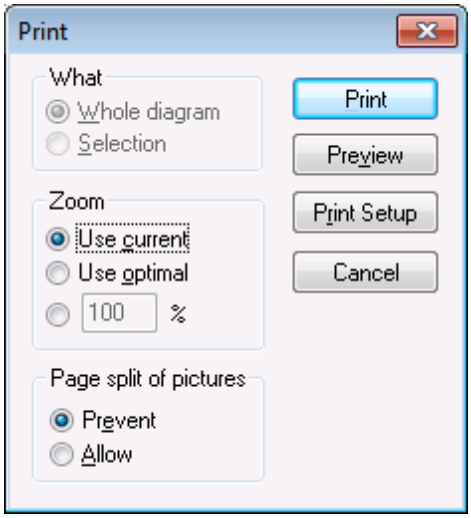
Save All
Saves all currently open mapping files.

Reload
Reloads the currently active mapping file. You are asked if you want to lose your last changes.

Close
Closes the currently active mapping file. You are asked if you want to save the file before it closes.

Close All
Closes all currently open mapping files. You are asked if you want to save any of the unsaved mapping files.

Print
Opens the Print dialog box, from where you can print out your mapping as hard copy.



Print dialog box

Use current retains the currently defined zoom factor of the mapping. **Use optimal** scales the mapping to fit the page size. You can also specify the zoom factor numerically. Component scrollbars are not printed. You can also specify if you want to allow the graphics to be split over several pages or not.

Print Preview
Opens the same Print dialog box with the same settings as described above.

Print Setup
Opens the Print Setup dialog box in which you can define the printer you want to use and the paper settings.

Validate Mapping
Validates that all mappings (connectors) are valid and displays any warnings or errors (see Validating mappings).

Mapping settings
Opens the Mapping Settings dialog box where you can define the document-specific settings (see Changing the mapping settings).

Generate code in selected language
Generates code in the currently selected language of your mapping. The currently selected language is visible as a highlighted programming language icon in the toolbar: XSLT, XSLT 2.

Generate code in | XSLT (XSLT2)
This command generates the XSLT file(s) needed for the transformation from the source file(s). Selecting this option opens the Browse for Folder dialog box where you select the location of the XSLT file. The name of the generated XSLT file(s) is defined in the Mapping Settings dialog box (see Changing the mapping settings).

Recent files
Displays a list of the most recently opened files.

 © 2015 Altova GmbH

Exit

Exits the application. You are asked if you want to save any unsaved files.

10.2 Edit

Most of the commands in this menu become active when you view the result of a mapping in the **Output** tab, or preview XSLT code in the XSLT tab.

Undo

MapForce has an unlimited number of "Undo" steps that you can use to retrace you mapping steps.

Redo

The redo command allows you to redo previously undone commands. You can step backward and forward through the undo history using both these commands.

Find

Allows you to search for specific text in either the XSLT, XSLT2 or Output tab.

Find Next F3

Searches for the next occurrence of the same search string.

Find Previous Shift F3

Searches for the previous occurrence of the same search string.

Cut/Copy/Paste/Delete

The standard windows Edit commands, allow you to cut, copy etc., any components or functions visible in the mapping window.

Select all

Selects all components in the Mapping tab, or the text/code in the XSLT, XSLT2, or Output tab.

© 2015 Altova GmbH

10.3 Insert

XML Schema / File

Inserts an XML schema file into the mapping tab as data source or target component. You can select XML files with a schema reference, in this case the referenced schema is automatically inserted. If you select an XML schema file, you are prompted if you want to include an XML instance file which supplies the data for the XSLT, XSLT2, , and Output previews. If you select an XML file without a schema reference, you are prompted if you want to generate a matching XML schema automatically.

Insert Input

When the mapping window displays a mapping, this command adds an input component to the mapping (see Simple Input). When the mapping window displays a user-defined function, this command adds an input component to the user-defined function (see Defining Complex Input Components).

Insert Output

When the mapping window displays a mapping, this command adds an output component to the mapping (see Simple Output). When the mapping window displays a user-defined function, this command adds an output component to the user-defined function (see Defining Complex Output Components).

Constant

Inserts a constant which is a function component that supplies fixed data to an input icon. The data is entered into a dialog box when creating the component. There is only one output icon on a constant function. You can select the following types of data: String, Number and All other.

Variable

Inserts an Intermediate Variable which is equivalent to a regular (non-inline) user-defined function. Variables are structural components, without instance files, and are used to simplify the mapping process. Please see Intermediate variables for more information.

Sort: Nodes/Rows

Inserts a component which allows you to sort nodes (see Sort Nodes/Rows).

Filter: Nodes/Rows

Inserts a component that uses two input and output parameters: **node/row** and **bool**, and **on-true**, **on-false**. If the Boolean is true, then the value of the node/row parameter is forwarded to the on-true parameter. If the Boolean is false, then the complement value is passed on to the on-false parameter. For more information, see Filtering.

Value-Map

Inserts a component that transforms an input value to an output value using a lookup table. The component only has one input and output item. Please see Value-Map - transforming input data for more information.

IF-Else Condition

Inserts a component of type "If-Else Condition" (see If-Else Condition).

© 2015 Altova GmbH

10.4 Component

Change Root Element
Allows you to change the root element of the XML instance document.

Edit Schema Definition in XMLSpy
Selecting this option, having previously clicked an XML-Schema/document, opens the XML Schema file in the Schema view of XMLSpy where you can edit it.

Add Duplicate Input Before
Inserts a copy/clone of the selected item before the currently selected item. Duplicate items do not have output icons, you cannot use them as data sources. For an example, see Map Multiple Sources to One Target section in the tutorial. Right clicking a duplicate item also allows you to reposition it using the menu items Move Up/Move Down, depending on where the item is.

Add Duplicate Input After
Inserts a copy/clone of the selected item after the currently selected item. Duplicate items do not have output icons, you cannot use them as data sources. For an example, see the Map Multiple Sources to One Target section in the tutorial. Right clicking a duplicate item also allows you to reposition it using the menu items Move Up/Move Down, depending on where the item is.

Remove Duplicate
Removes a previously defined duplicate item. For an example, see the Map Multiple Sources to One Target section in the tutorial.

Align Tree Left
Aligns all the items along the left hand window border.

Align Tree Right
Aligns all the items along the right hand window border. This display is useful when creating mappings to the target schema.

Properties
Opens a dialog box which displays the settings of the currently selected component. See Changing the Component Settings .

10.5 Connection

Auto Connect Matching Children
Activates or deactivates the "Auto Connect Matching Children" option, as well as the icon in the icon bar.

Settings for Connect Matching Children
Opens the Connect Matching Children dialog box in which you define the connection settings (see Connecting matching children).

Connect Matching Children
This command allows you to create multiple connectors for items of the **same name,** in both the source and target schemas. The settings you define in this dialog box are retained, and are applied when connecting two items, if the "**Auto connect child items**" icon in the title bar is active. Clicking the icon switches between an active and inactive state. For further information, see Connecting matching children.

Target Driven (Standard)
Changes the connector type to Standard mapping. For further information, see Target Driven (Standard) mapping.

Copy-all (Copy Child Items)
Creates connectors for all matching child items, where each of the child connectors are displayed as a subtree of the parent connector (see Copy-all connections).

Source Driven (Mixed Content)
Changes the connector type to *Source Driven (Mixed Content)*. For further information, see Source Driven (Mixed Content) mapping.

Properties
Opens a dialog box in which you can define the specific (mixed content) settings of the current connector. Unavailable options are greyed out. These settings also apply to **complexType** items which do not have any text nodes. For further information, see Connection settings.

© 2015 Altova GmbH

10.6 Function

Create User-Defined Function
Creates a new user-defined function (see User-defined functions).

Create User-Defined Function from Selection
Creates a new user-defined function based on the currently selected elements in the mapping window.

Function Settings
Opens the settings dialog box of the currently active user-defined function allowing you to change its settings.

Remove Function
Deletes the currently active user-defined function if you are working in a context which allows this.

Insert Input
When the mapping window displays a mapping, this command adds an input component to the mapping (see Simple Input). When the mapping window displays a user-defined function, this command adds an input component to the user-defined function (see Defining Complex Input Components).

Insert Output
When the mapping window displays a mapping, this command adds an output component to the mapping (see Simple Output). When the mapping window displays a user-defined function, this command adds an output component to the user-defined function (see Defining Complex Output Components).

10.7 Output

The first group of options (**XSLT 1.0, XSLT 2.0, etc.)** allow you to define the target language you want your code to be in.

Validate Output
Validates the output XML file against the referenced schema.

Save generated Output
Saves the currently visible data in the Output tab.

Save all generated outputs
Saves all the generated output files of dynamic mappings. See Processing Multiple Input or Output Files Dynamically for more information.

Regenerate output
Regenerates the current mapping from the Output window.

Insert/Remove Bookmark
Inserts a bookmark at the cursor position in the Output window.

Next Bookmark
Navigates to the next bookmark in the Output window.

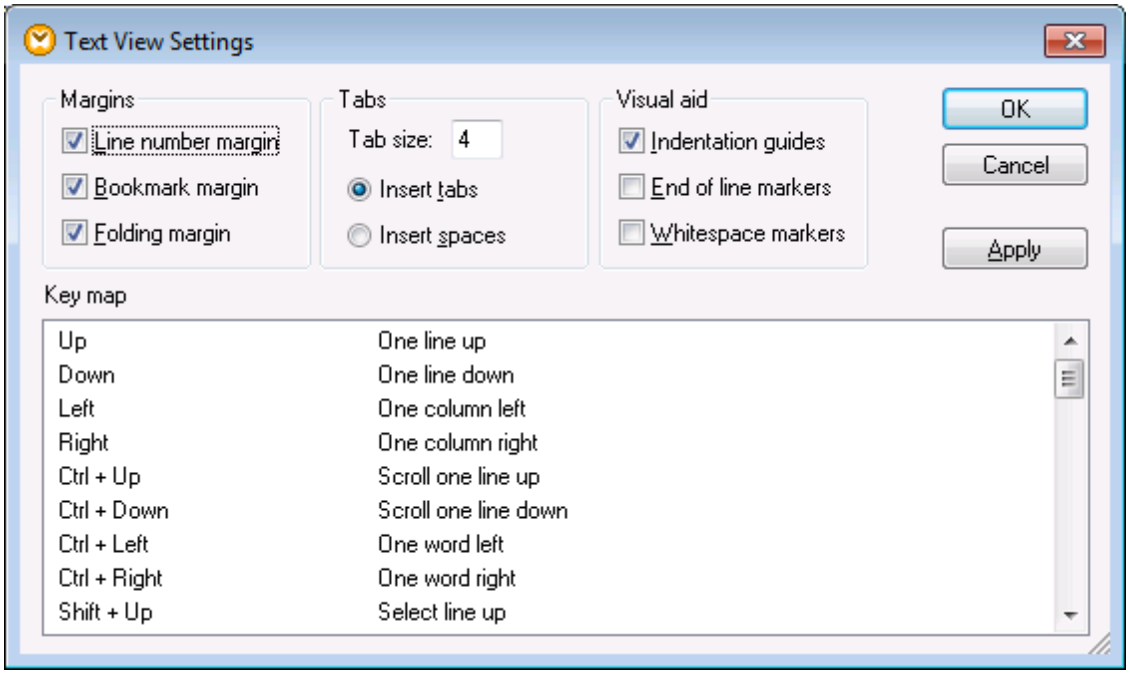
Previous Bookmark
Navigates to the previous bookmark in the Output window.

Remove All Bookmarks
Removes all currently defined bookmarks in the Output window.

Pretty-Print XML Text
Reformats your XML document in the Output pane to give a structured display of the document. Each child node is offset from its parent by a single tab character. This is where the Tab size settings (i.e. inserting as tabs or spaces) defined in the Tabs group, take effect.

Text View Settings
Allows you to customize the text settings in the Output window and also shows the currently defined hotkeys that apply in the window.

© 2015 Altova GmbH

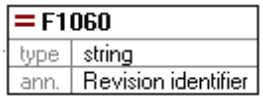


Note: The **Insert tabs** and **Insert spaces** options do not affect the currently visible text in the Output window; they only take effect when you use the **Output | Pretty-Print XML text** option.

10.8 View

Show Annotations

Displays XML schema annotations in the component window.
If the Show Types icon is also active, then both sets of info are show in grid form.



Show Types

Displays the schema datatypes for each element or attribute.
If the Show Annotations icon is also active, then both sets of info are show in grid form.

Show library in Function Header

Displays the library name in parenthesis in the function title.

Show Tips

Displays a tooltip containing explanatory text when the mouse pointer is placed over a function.

Show Selected Component Connectors

Switches between showing all mapping connectors, or those connectors relating to the currently selected components.

Show Connectors from Source to Target

Switches between showing:

- connectors that are **directly** connected to the currently selected component, or
- connectors linked to the currently selected component, originating from source and terminating at the target components.

Zoom

Opens the Zoom dialog box. You can enter the zoom factor numerically, or drag the slider to change the zoom factor interactively.

Back

Steps back through the currently open mappings of the mapping tab.

Forward

Steps forward through the currently open mappings of the mapping tab.

Status Bar

Switches on/off the Status Bar visible below the Messages window.

Library Window

Switches on/off the Library window.

Messages

Switches on/off the Validation output window. When generating code the Messages output

 © 2015 Altova GmbH

window is automatically activated to show the validation result.

Overview

Switches on/off the Overview window. Drag the rectangle to navigate your Mapping view.

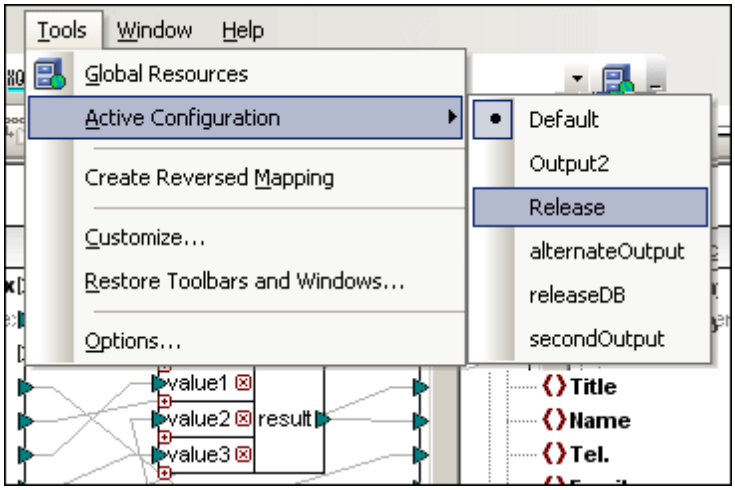
10.9 Tools

Global Resources

Opens the Manage Global Resources dialog box allowing you to Add, Edit and Delete global resources to the Global Resources XML file, please see Global Resources - Properties for more information.

Active Configuration

Allows you to select/change the currently active global resource from a list of all resources/ configurations in the Global Resources. Select the required configuration from the submenu.

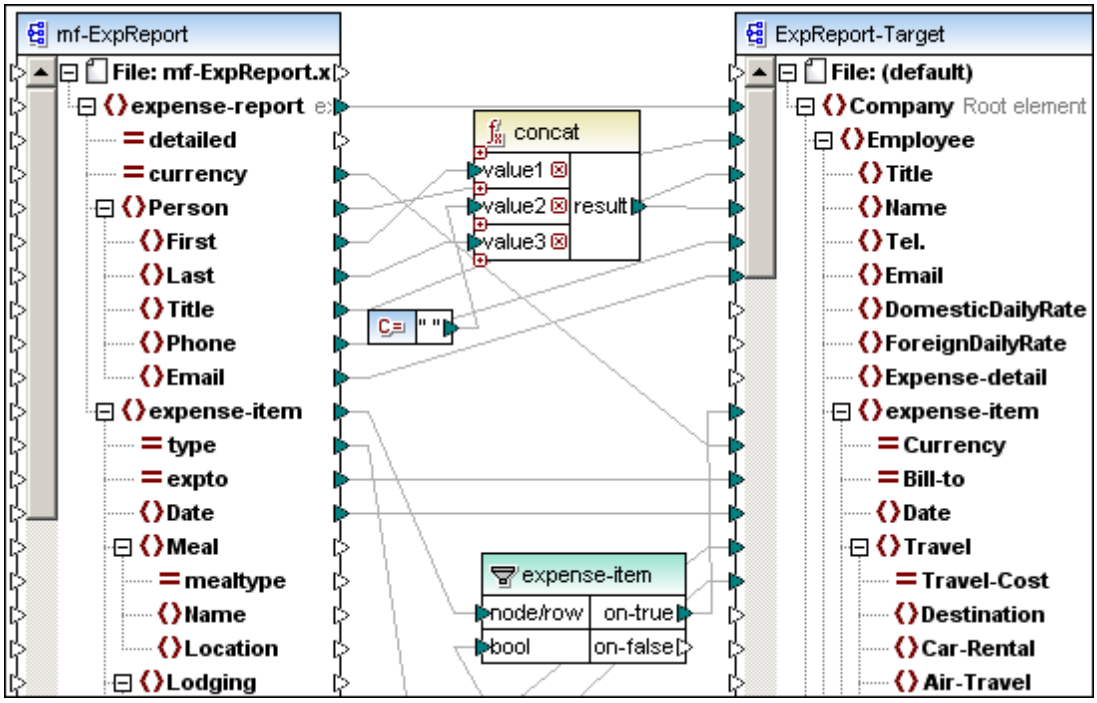


Create Reversed Mapping

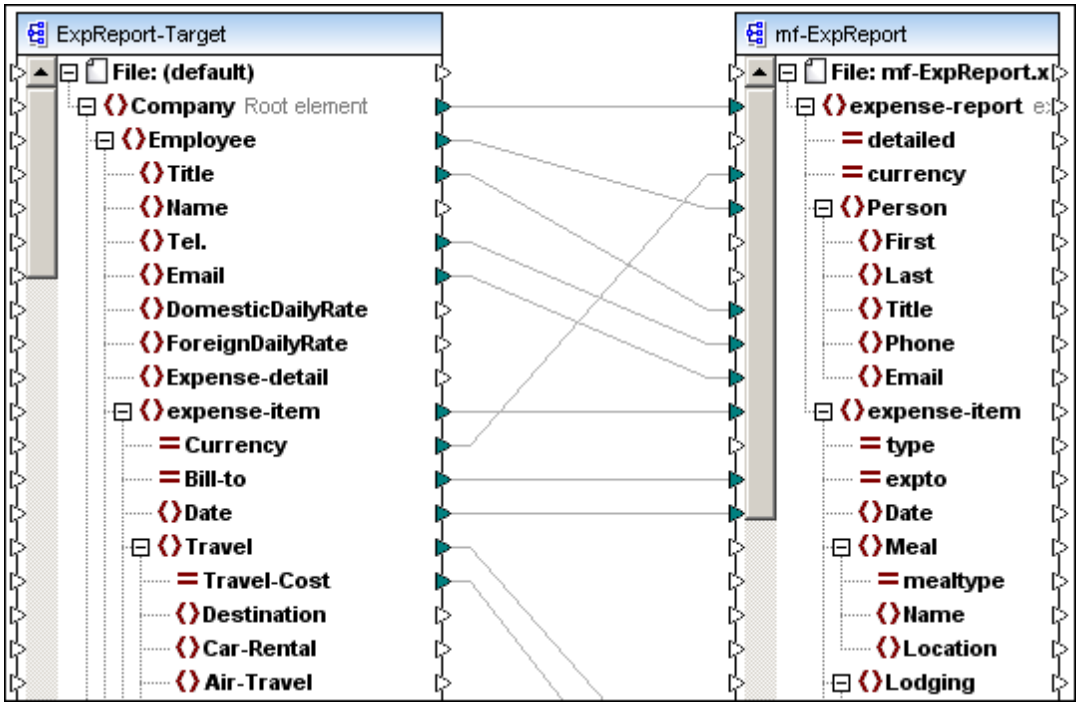
Creates a "reversed" mapping from the currently active mapping in MapForce, which is to be the basis of a new mapping. Note that the result is not intended to be a complete mapping, only the direct connections between components are retained in the reversed mapping. It is very likely that the resulting mapping will not be valid, or be able to be executed when clicking the Output tab, without manual editing.

E.g. **Tut-ExpReport.mfd** in the ...\MapForceExamples\Tutorial folder:

© 2015 Altova GmbH



Result of a reversed mapping:



General:

- The source component becomes the target component, and target component becomes the source.
- If an Input, and Output XML, instance file have been assigned to a component, then they will both be swapped.

Retained connections

- Direct connections between components
- Direct connections between components in a chained mapping
- The type of connection: Standard, Mixed content, Copy-All

- Pass-through component settings
- Database components are unchanged.

Deleted connections

- Connections via functions, filters etc. are deleted, along with the functions etc.
- User-defined functions
- Webservice components

Restore Toolbars and Windows

Resets the toolbars, entry helper windows, docked windows etc. to their defaults. MapForce needs to be restarted for the changes to take effect.

Customize...

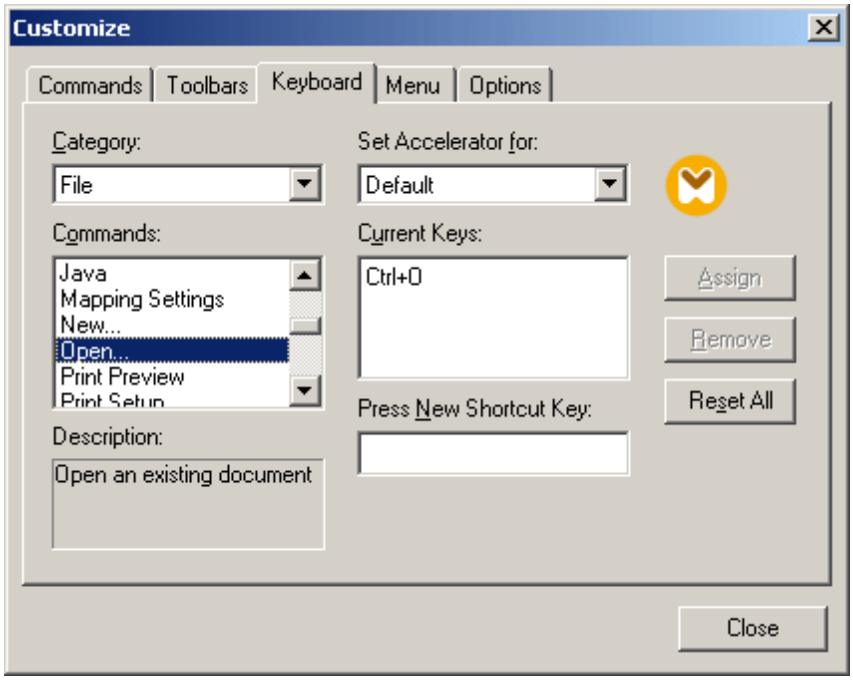
The customize command lets you customize MapForce to suit your personal needs.

The **Keyboard** tab allows you to define (or change) keyboard shortcuts for any MapForce command.

To assign a new Shortcut to a command:

1. Select the **Tools | Customize** command and click the Keyboard tab.
2. Click the **Category** combo box to select the menu name.
3. Select the **command** you want to assign a new shortcut to, in the Commands list box
4. Click in the **Press New Shortcut Key**: text box, and press the shortcut keys that are to activate the command.

© 2015 Altova GmbH



The shortcuts appear immediately in the text box. If the shortcut was assigned previously, then that function is displayed below the text box.

5. Click the **Assign** button to assign the shortcut.
   The shortcut now appears in the Current Keys list box.
   (To **clear** the entry in the Press New Shrotcut Key text box, press any of the control keys, **CTRL**, **ALT** or **SHIFT**).

To de-assign or delete a shortcut:

1. Click the shortcut you want to delete in the Current Keys list box.
2. Click the **Remove** button.
3. Click the **Close** button to confirm.

Set accelerator for:

Currently no function.

Currently assigned keyboard shortcuts:

Hotkeys by key

F1	Help Menu
F2	Next bookmark (in output window)
F3	Find Next
F10	Activate menu bar
Num +	Expand current item node
Num -	Collapse item node
Num *	Expand all from current item node
CTRL + TAB	Switches between open mappings
CTRL + F6	Cycle through open windows
CTRL + F4	Closes the active mapping document
Alt + F4	Closes MapForce
Alt + F, F, 1	Opens the last file
Alt + F, T, 1	Opens the last project

CTRL + N	File New
CTRL + O	File Open
CTRL + S	File Save
CTRL + P	File Print
CTRL + A	Select All
CTRL + X	Cut
CTRL + C	Copy
CTRL + V	Paste
CTRL + Z	Undo
CTRL + Y	Redo
Del	Delete component (with prompt)
Shift + Del	Delete component (no prompt)
CTRL + F	Find
F3	Find Next
Shift + F3	Find Previous
Arrow keys	
(up / down)	Select next item of component
Esc	Abandon edits/close dialog box
Return	Confirms a selection
Output window hotkeys	
CTRL + F2	Insert Remove/Bookmark
F2	Next Bookmark
SHIFT + F2	Previous Bookmark
CTRL + SHIFT + F2	Remove All Bookmarks
Zooming hotkeys	
CTRL + mouse wheel forward	Zoom In
CTRL + mouse wheel back	Zoom Out
CTRL + 0 (Zero)	Reset Zoom

Options

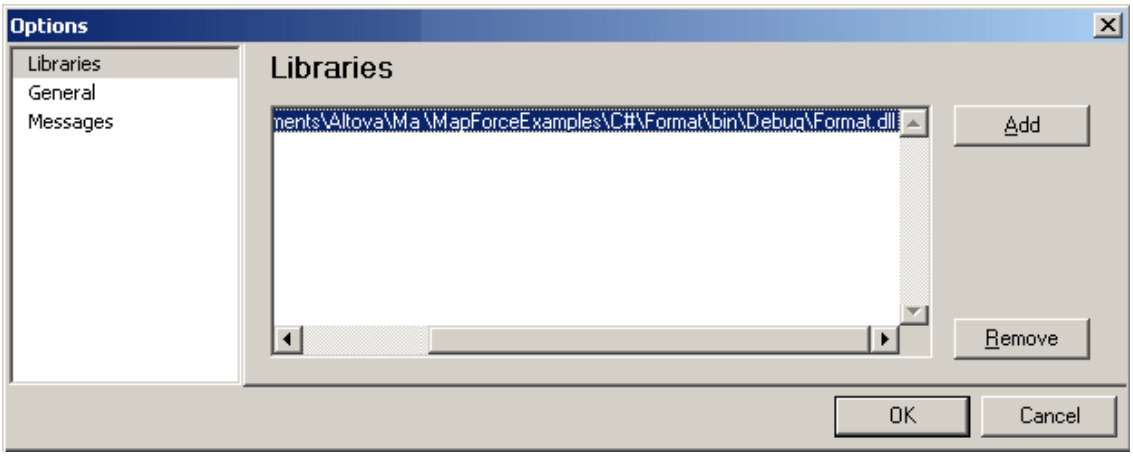
Opens the Options dialog box through which you can:

- Add or delete user defined XSLT functions.
- Define **general** settings, such as the default character encoding for new components, in the General tab.
- Define which message notifications you want to re-enable

Libraries tab:

- Add or delete user-defined XSLT, or programming language Libraries/functions to MapForce.

© 2015 Altova GmbH

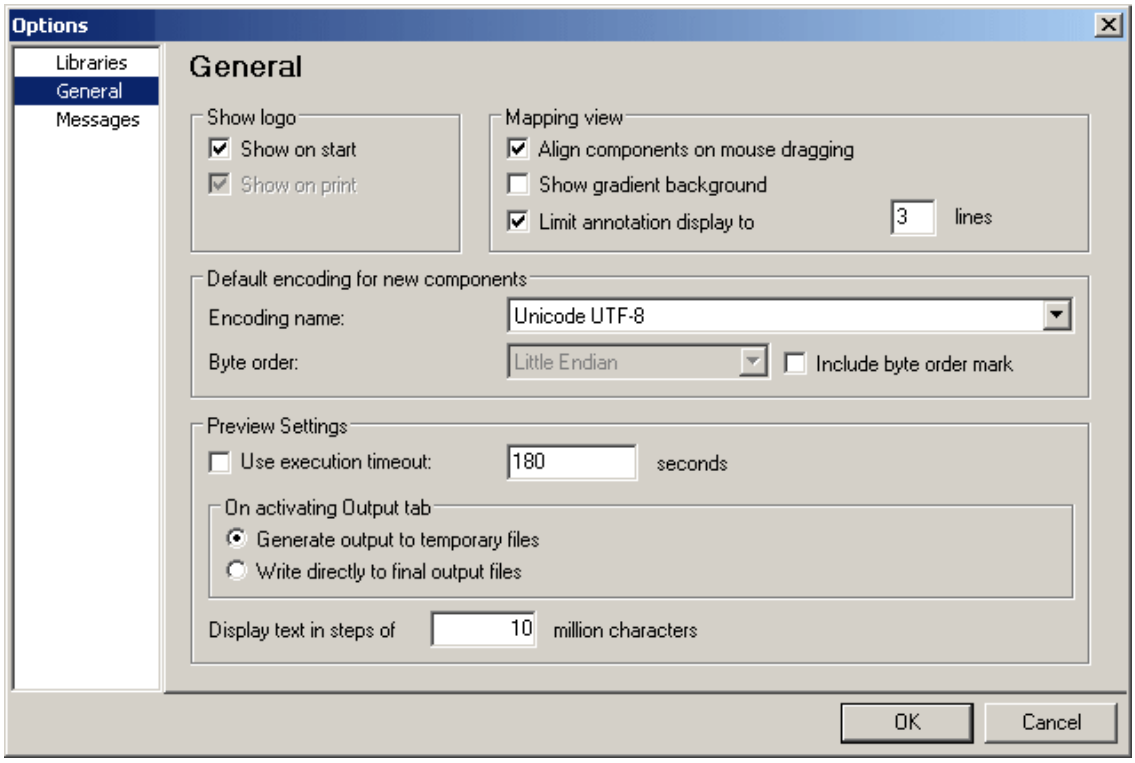


General tab:

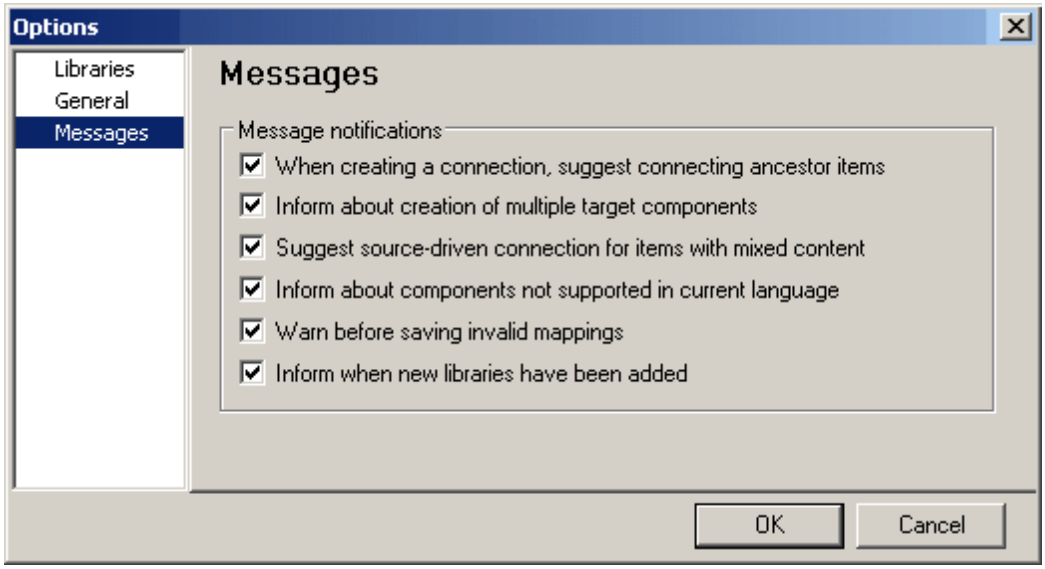
- Specify if you want to show the logo, copyright etc., on start and/or when printing.
- Align components or functions with other components, while dragging them with the mouse.
- Enable/disable the MapForce gradient background.
- Limit the annotation text in components to X lines. Also applies to SELECT statements visible in a component.
- Define the default character encoding for new components.
- Define an execution timeout for the Output tab when previewing the mapping result.
- Specify if you want to output to **temporary** files (default), or write output files **directly** to disk when clicking the Output button/tab.

  Warning: Enabling "Write directly to final output files" will overwrite output files without requesting further confirmation.

- Limit the output to X million characters, when outputting to the built-in execution engine. The Built-in execution engine is the only target that supports XML, CSV, and FLF streaming



Messages tab:
Allows you to re-enable message boxes that you previously disabled using the "Don't ask me again" check box.



 © 2015 Altova GmbH

10.10 Window

Cascade

This command rearranges all open document windows so that they are all cascaded (i.e. staggered) on top of each other.

Tile Horizontal

This command rearranges all open document windows as **horizontal tiles**, making them all visible at the same time.

Tile Vertical

This command rearranges all open document windows as **vertical tiles**, making them all visible at the same time.

1
2
This list shows all currently open windows, and lets you quickly switch between them. You can also use the Ctrl-TAB or CTRL F6 keyboard shortcuts to cycle through the open windows.

10.11 Help Menu

The **Help** menu contains commands to access the onscreen help manual for MapForce, commands to provide information about MapForce, and links to support pages on the Altova web site. The Help menu also contains the Registration dialog, which lets you enter your license key-code once you have purchased the product.

The description of the Help menu commands is organized into the following sub-sections:

- Table of Contents, Index, Search
- Registration, Order Form
- Other Commands

10.11.1 Table of Contents, Index, Search

The **Table of Contents** command opens the onscreen help manual for MapForce with the Table of Contents displayed in the left-hand-side pane of the Help window. The Table of Contents provides a good overview of the entire Help document. Clicking an entry in the Table of Contents takes you to that topic.

The **Index** command opens the onscreen help manual for MapForce with the Keyword Index displayed in the left-hand-side pane of the Help window. The index lists keywords and lets you navigate to a topic by double-clicking the keyword. If a keyword is linked to more than one topic, you are presented with a list of the topics to choose from.

The **Search** command opens the onscreen help manual for MapForce with the Search dialog displayed in the left-hand-side pane of the Help window. To search for a term, enter the term in the input field, and press Return. The Help system performs a full-text search on the entire Help documentation and returns a list of hits. Double-click any item to display that item.

10.11.2 Activation, Order Form, Registration, Updates

Software Activation

After you download your Altova product software, you can activate it using either a free evaluation key or a purchased permanent license key.

- ***Free evaluation key.*** When you first start the software after downloading and installing it, the Software Activation dialog will pop up. In it is a button to request a free evaluation key-code. Enter your name, company, and e-mail address in the dialog that appears, and click Request Now! The evaluation key is sent to the e-mail address you entered and should reach you in a few minutes. Now enter the key in the key-code field of the Software Activation dialog box and click **OK** to start working with your Altova product. The software will be unlocked for a period of 30 days.
- ***Permanent license key.*** The Software Activation dialog contains a button to purchase a permanent license key. Clicking this button takes you to Altova's online shop, where you can purchase a permanent license key for your product. There are two types of permanent license: single-user and multi-user. Both will be sent to you by e-mail. A *single-user license* contains your license-data and includes your name, company, e-mail, and key-code.A *multi-user license* contains your license-data and includes your company name and key-code. Note that your license agreement does not allow you to install more

 © 2015 Altova GmbH

than the licensed number of copies of your Altova software on the computers in your organization (per-seat license). Please make sure that you enter the data required in the registration dialog exactly as given in your license e-mail.

Note: When you enter your license information in the Software Activation dialog, ensure that you enter the data exactly as given in your license e-mail. For multi-user licenses, each user should enter his or her own name in the Name field.

The Software Activation dialog can be accessed at any time by clicking the **Help | Software Activation** command.

Order Form

When you are ready to order a licensed version of MapForce, you can use either the **Order license key** button in the Software Activation dialog (*see previous section*) or the **Help | Order Form** command to proceed to the secure Altova Online Shop.

Registration

The first time you start your Altova software after having activated it, a dialog appears asking whether you would like to register your product. There are three buttons in this dialog:

- **OK:** Takes you to the Registration Form
- **Remind Me Later:** Pops up a dialog in which you can select when you wish to be next reminded.
- **Cancel:** Closes the dialog and suppresses it in future. If you wish to register at a later time, you can use the **Help | Registration** command.

Check for Updates

Checks with the Altova server whether a newer version than yours is currently available and displays a message accordingly

10.11.3 Other Commands

The **Support Center** command is a link to the Altova Support Center on the Internet. The Support Center provides FAQs, discussion forums where problems are discussed, and access to Altova's technical support staff.

The **FAQ on the Web** command is a link to Altova's FAQ database on the Internet. The FAQ database is constantly updated as Altova support staff encounter new issues raised by customers.

The **Components Download** command is a link to Altova's Component Download Center on the Internet. From here you will be able to download a variety of companion software to use with Altova products. Such software ranges from XSLT and XSL-FO processors to Application Server Platforms. The software available at the Component Download Center is typically free of charge.

The **MapForce on the Internet** command is a link to the Altova website on the Internet. You can learn more about MapForce and related technologies and products at the Altova website.

The **MapForce Training** command is a link to the Online Training page at the Altova website.

Here you can select from online courses conducted by Altova's expert trainers.

The **About MapForce** command displays the splash window and version number of your product.

© 2015 Altova GmbH

Chapter 11

Appendices

11 Appendices

These appendices contain technical information about MapForce and important licensing information. Each appendix contains sub-sections as given below:

Technical Data

- OS and memory requirements
- Altova XML Parser
- Altova XSLT and XQuery Engines
- Unicode support
- Internet usage
- License metering

License Information

- Electronic software distribution
- Copyrights
- End User License Agreement

© 2015 Altova GmbH

11.1 Engine information

This section contains information about implementation-specific features of the Altova XML Validator, Altova XSLT 1.0 Engine, Altova XSLT 2.0 Engine, and Altova XQuery Engine.

11.1.1 XSLT and XQuery Engine Information

The XSLT and XQuery engines of MapForce follow the W3C specifications closely and are therefore stricter than previous Altova engines—such as those in previous versions of XMLSpy. As a result, minor errors that were ignored by previous engines are now flagged as errors by MapForce.

For example:

- It is a type error (`err:XPTY0018`) if the result of a path operator contains both nodes and non-nodes.
- It is a type error (`err:XPTY0019`) if `E1` in a path expression `E1/E2` does not evaluate to a sequence of nodes.

If you encounter this kind of error, modify either the XSLT/XQuery document or the instance document as appropriate.

This section describes implementation-specific features of the engines, organized by specification:

- XSLT 1.0
- XSLT 2.0
- XQuery 1.0

11.1.1.1 XSLT 1.0

The XSLT 1.0 Engine of MapForce conforms to the World Wide Web Consortium's (W3C's) XSLT 1.0 Recommendation of 16 November 1999 and XPath 1.0 Recommendation of 16 November 1999. Note the following information about the implementation.

Notes about the implementation

When the `method` attribute of `xsl:output` is set to HTML, or if HTML output is selected by default, then special characters in the XML or XSLT file are inserted in the HTML document as HTML character references in the output. For instance, the character ` ` (the decimal character reference for a non-breaking space) is inserted as ` ` in the HTML code.

11.1.1.2 XSLT 2.0

This section:

- Engine conformance

- Backward compatibility
- Namespaces
- Schema awareness
- Implementation-specific behavior

Conformance

The XSLT 2.0 engine of MapForce conforms to the World Wide Web Consortium's (W3C's) XSLT 2.0 Recommendation of 23 January 2007 and XPath 2.0 Recommendation of 14 December 2010.

Backwards Compatibility

The XSLT 2.0 engine is backwards compatible. The only time the backwards compatibility of the XSLT 2.0 engine comes into effect is when using the XSLT 2.0 engine to process an XSLT 1.0 stylesheet. Note that there could be differences in the outputs produced by the XSLT 1.0 Engine and the backwards-compatible XSLT 2.0 engine.

Namespaces

Your XSLT 2.0 stylesheet should declare the following namespaces in order for you to be able to use the type constructors and functions available in XSLT 2.0. The prefixes given below are conventionally used; you could use alternative prefixes if you wish.

Namespace Name	Prefix	Namespace URI
XML Schema types	`xs:`	`http://www.w3.org/2001/XMLSchema`
XPath 2.0 functions	`fn:`	`http://www.w3.org/2005/xpath-functions`

Typically, these namespaces will be declared on the `xsl:stylesheet` or `xsl:transform` element, as shown in the following listing:

```
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"
    ...
</xsl:stylesheet>
```

The following points should be noted:

- The XSLT 2.0 engine uses the XPath 2.0 and XQuery 1.0 Functions namespace (listed in the table above) as its **default functions namespace**. So you can use XPath 2.0 and XSLT 2.0 functions in your stylesheet without any prefix. If you declare the XPath 2.0 Functions namespace in your stylesheet with a prefix, then you can additionally use the prefix assigned in the declaration.
- When using type constructors and types from the XML Schema namespace, the prefix

 © 2015 Altova GmbH

used in the namespace declaration must be used when calling the type constructor (for example, `xs:date`).

- Some XPath 2.0 functions have the same name as XML Schema datatypes. For example, for the XPath functions `fn:string` and `fn:boolean` there exist XML Schema datatypes with the same local names: `xs:string` and `xs:boolean`. So if you were to use the XPath expression `string('Hello')`, the expression evaluates as `fn:string('Hello')`—not as `xs:string('Hello')`.

Schema-awareness
The XSLT 2.0 engine is schema-aware. So you can use user-defined schema types and the `xsl:validate` instruction.

Implementation-specific behavior
Given below is a description of how the XSLT 2.0 engine handles implementation-specific aspects of the behavior of certain XSLT 2.0 functions.

`xsl:result-document`
Additionally supported encodings are (the Altova-specific): `x-base16tobinary` and `x-base64tobinary`.

`function-available`
The function tests for the availability of in-scope functions (XSLT 2.0, XPath 2.0, and extension functions).

`unparsed-text`
The `href` attribute accepts (i) relative paths for files in the base-uri folder, and (ii) absolute paths with or without the `file://` protocol. Additionally supported encodings are (the Altova-specific): `x-binarytobase16` and `x-binarytobase64`.

`unparsed-text-available`
The `href` attribute accepts (i) relative paths for files in the base-uri folder, and (ii) absolute paths with or without the `file://` protocol. Additionally supported encodings are (the Altova-specific): `x-binarytobase16` and `x-binarytobase64`.

Note: The following encoding values, which were implemented in earlier versions of RaptorXML's predecessor product, AltovaXML, are now deprecated: `base16tobinary`, `base64tobinary`, `binarytobase16` and `binarytobase64`.

11.1.1.3 XQuery 1.0

This section:

- Engine conformance
- Schema awareness
- Encoding
- Namespaces
- XML source and validation
- Static and dynamic type checking
- Library modules
- External modules
- Collations
- Precision of numeric data
- XQuery instructions support

Conformance
The XQuery 1.0 Engine of MapForce conforms to the World Wide Web Consortium's (W3C's) XQuery 1.0 Recommendation of 14 December 2010. The XQuery standard gives implementations discretion about how to implement many features. Given below is a list explaining how the XQuery 1.0 Engine implements these features.

Schema awareness
The XQuery 1.0 Engine is **schema-aware**.

Encoding
The UTF-8 and UTF-16 character encodings are supported.

Namespaces
The following namespace URIs and their associated bindings are pre-defined.

Namespace Name	Prefix	Namespace URI
XML Schema types	`xs:`	`http://www.w3.org/2001/XMLSchema`
Schema instance	`xsi:`	`http://www.w3.org/2001/XMLSchema-instance`
Built-in functions	`fn:`	`http://www.w3.org/2005/xpath-functions`
Local functions	`local:`	`http://www.w3.org/2005/xquery-local-functions`

The following points should be noted:

- The XQuery 1.0 Engine recognizes the prefixes listed above as being bound to the corresponding namespaces.

 © 2015 Altova GmbH

- Since the built-in functions namespace listed above is the default functions namespace in XQuery, the `fn:` prefix does not need to be used when built-in functions are invoked (for example, `string("Hello")` will call the `fn:string` function). However, the prefix `fn:` can be used to call a built-in function without having to declare the namespace in the query prolog (for example: `fn:string("Hello")`).
- You can change the default functions namespace by declaring the `default function namespace` expression in the query prolog.
- When using types from the XML Schema namespace, the prefix `xs:` may be used without having to explicitly declare the namespaces and bind these prefixes to them in the query prolog. (Example: `xs:date` and `xs:yearMonthDuration`.) If you wish to use some other prefix for the XML Schema namespace, this must be explicitly declared in the query prolog. (Example: `declare namespace alt = "http://www.w3.org/2001/XMLSchema"; alt:date("2004-10-04")`.)
- Note that the `untypedAtomic`, `dayTimeDuration`, and `yearMonthDuration` datatypes have been moved, with the CRs of 23 January 2007, from the XPath Datatypes namespace to the XML Schema namespace, so: `xs:yearMonthDuration`.

If namespaces for functions, type constructors, node tests, etc are wrongly assigned, an error is reported. Note, however, that some functions have the same name as schema datatypes, e.g. `fn:string` and `fn:boolean`. (Both `xs:string` and `xs:boolean` are defined.) The namespace prefix determines whether the function or type constructor is used.

XML source document and validation

XML documents used in executing an XQuery document with the XQuery 1.0 Engine must be well-formed. However, they do not need to be valid according to an XML Schema. If the file is not valid, the invalid file is loaded without schema information. If the XML file is associated with an external schema and is valid according to it, then post-schema validation information is generated for the XML data and will be used for query evaluation.

Static and dynamic type checking

The static analysis phase checks aspects of the query such as syntax, whether external references (e.g. for modules) exist, whether invoked functions and variables are defined, and so on. If an error is detected in the static analysis phase, it is reported and the execution is stopped.

Dynamic type checking is carried out at run-time, when the query is actually executed. If a type is incompatible with the requirement of an operation, an error is reported. For example, the expression `xs:string("1") + 1` returns an error because the addition operation cannot be carried out on an operand of type `xs:string`.

Library Modules

Library modules store functions and variables so they can be reused. The XQuery 1.0 Engine

supports modules that are stored in **a single external XQuery file**. Such a module file must contain a `module` declaration in its prolog, which associates a target namespace. Here is an example module:

```
module namespace libns="urn:module-library";
declare variable $libns:company := "Altova";
declare function libns:webaddress() { "http://www.altova.com" };
```

All functions and variables declared in the module belong to the namespace associated with the module. The module is used by importing it into an XQuery file with the `import module` statement in the query prolog. The `import module` statement only imports functions and variables declared directly in the library module file. As follows:

```
import module namespace modlib = "urn:module-library" at "modulefilename.xq";

if        ($modlib:company = "Altova")
then       modlib:webaddress()
else       error("No match found.")
```

External functions

External functions are not supported, i.e. in those expressions using the `external` keyword, as in:

```
declare function hoo($param as xs:integer) as xs:string external;
```

Collations

The default collation is the Unicode-codepoint collation, which compares strings on the basis of their Unicode codepoint. Other supported collations are the ICU collations listed here. To use a specific collation, supply its URI as given in the list of supported collations. Any string comparisons, including for the `fn:max` and `fn:min` functions, will be made according to the specified collation. If the collation option is not specified, the default Unicode-codepoint collation is used.

Precision of numeric types

- The `xs:integer` datatype is arbitrary-precision, i.e. it can represent any number of digits.
- The `xs:decimal` datatype has a limit of 20 digits after the decimal point.
- The `xs:float` and `xs:double` datatypes have limited-precision of 15 digits.

XQuery Instructions Support

The `Pragma` instruction is not supported. If encountered, it is ignored and the fallback expression is evaluated.

 © 2015 Altova GmbH

11.1.2 XSLT and XPath/XQuery Functions

This section lists Altova extension functions and other extension functions that can be used in XPath and/or XQuery expressions. Altova extension functions can be used with Altova's XSLT and XQuery engines, and provide functionality additional to that available in the function libraries defined in the W3C standards.

General points

The following general points should be noted:

- Functions from the core function libraries defined in the W3C specifications can be called without a prefix. That's because the XSLT and XQuery engines read non-prefixed functions as belonging to a default functions namespace which is that specified in the XPath/XQuery functions specifications`http://www.w3.org/2005/xpath-functions`. If this namespace is explicitly declared in an XSLT or XQuery document, the prefix used in the namespace declaration can also optionally be used on function names.
- In general, if a function expects a sequence of one item as an argument, and a sequence of more than one item is submitted, then an error is returned.
- All string comparisons are done using the Unicode codepoint collation.
- Results that are QNames are serialized in the form `[prefix:]localname`.

Precision of xs:decimal

The precision refers to the number of digits in the number, and a minimum of 18 digits is required by the specification. For division operations that produce a result of type `xs:decimal`, the precision is 19 digits after the decimal point with no rounding.

Implicit timezone

When two `date`, `time`, or `dateTime` values need to be compared, the timezone of the values being compared need to be known. When the timezone is not explicitly given in such a value, the implicit timezone is used. The implicit timezone is taken from the system clock, and its value can be checked with the `implicit-timezone()` function.

Collations

The default collation is the Unicode codepoint collation, which compares strings on the basis of their Unicode codepoint. Other supported collations are the ICU collations listed below. To use a specific collation, supply its URI as given in the list of supported collations (*table below*). Any string comparisons, including for the `max` and `min` functions, will be made according to the specified collation. If the collation option is not specified, the default Unicode-codepoint collation is used.

Language	URIs
`da`: Danish	`da_DK`

de: German	de_AT, de_BE, de_CH, de_DE, de_LI, de_LU
en: English	en_AS, en_AU, en_BB, en_BE, en_BM, en_BW, en_BZ, en_CA, en_GB, en_GU, en_HK, en_IE, en_IN, en_JM, en_MH, en_MP, en_MT, en_MU, en_NA, en_NZ, en_PH, en_PK, en_SG, en_TT, en_UM, en_US, en_VI, en_ZA, en_ZW
es: Spanish	es_419, es_AR, es_BO, es_CL, es_CO, es_CR, es_DO, es_EC, es_ES, es_GQ, es_GT, es_HN, es_MX, es_NI, es_PA, es_PE, es_PR, es_PY, es_SV, es_US, es_UY, es_VE
fr: French	fr_BE, fr_BF, fr_BI, fr_BJ, fr_BL, fr_CA, fr_CD, fr_CF, fr_CG, fr_CH, fr_CI, fr_CM, fr_DJ, fr_FR, fr_GA, fr_GN, fr_GP, fr_GQ, fr_KM, fr_LU, fr_MC, fr_MF, fr_MG, fr_ML, fr_MQ, fr_NE, fr_RE, fr_RW, fr_SN, fr_TD, fr_TG
it: Italian	it_CH, it_IT
ja: Japanese	ja_JP
nb: Norwegian Bokmal	nb_NO
nl: Dutch	nl_AW, nl_BE, nl_NL
nn: Nynorsk	nn_NO
pt: Portuguese	pt_AO, pt_BR, pt_GW, pt_MZ, pt_PT, pt_ST
ru: Russian	ru_MD, ru_RU, ru_UA
sv: Swedish	sv_FI, sv_SE

Namespace axis
The namespace axis is deprecated in XPath 2.0. Use of the namespace axis is, however, supported. To access namespace information with XPath 2.0 mechanisms, use the `in-scope-prefixes()`, `namespace-uri()` and `namespace-uri-for-prefix()` functions.

11.1.2.1 Altova Extension Functions

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

Functions defined in the W3C's XPath/XQuery Functions specifications can be used in: (i) XPath expressions in an XSLT context, and (ii) in XQuery expressions in an XQuery document. In this documentation we indicate the functions that can be used in the former context (XPath in XSLT) with an **`XP`** symbol and call them XPath functions; those functions that can be used in the latter

(XQuery) context are indicated with an XQ symbol; they work as XQuery functions. The W3C's XSLT specifications—not XPath/XQuery Functions specifications—also define functions that can be used in XPath expressions in XSLT documents. These functions are marked with an XSLT symbol and are called XSLT functions. The XPath/XQuery and XSLT versions in which a function can be used are indicated in the description of the function (*see symbols below*). Functions from the XPath/XQuery and XSLT function libraries are listed without a prefix. Extension functions from other libraries, such as Altova extension functions, are listed with a prefix.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

XSLT functions

XSLT functions can only be used in XPath expressions in an XSLT context (similarly to XSLT 2.0's `current-group()` or `key()` functions). These functions are not intended for, and will not work in, a non-XSLT context (for instance, in an XQuery context). Note that XSLT functions for XBRL can be used only with editions of Altova products that have XBRL support.

XPath/XQuery functions

XPath/XQuery functions can be used both in XPath expressions in XSLT contexts as well as in XQuery expressions:

- Date/Time
- Geolocation
- Image-related
- Numeric
- Sequence
- String
- Miscellaneous

XSLT Functions

XSLT extension functions can be used in XPath expressions in an XSLT context. They will not work in a non-XSLT context (for instance, in an XQuery context).

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace,** `http://www.altova.com/xslt-extensions`, and are indicated in this section with the prefix `altova:`, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information

about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

Standard functions

distinct-nodes [altova:]

altova:distinct-nodes(*node()**)** as **node()*** XSLT1 XSLT2 XSLT3

Takes a set of one or more nodes as its input and returns the same set minus nodes with duplicate values. The comparison is done using the XPath/XQuery function `fn:deep-equal`.

Examples

- **altova:distinct-nodes**(`country`) returns all child `country` nodes less those having duplicate values.

evaluate [altova:]

altova:evaluate(XPathExpression *as xs:string***[, ValueOf$p1, ... ValueOf$pN])** XSLT1 XSLT2 XSLT3

Takes an XPath expression, passed as a string, as its mandatory argument. It returns the output of the evaluated expression. For example: **altova:evaluate('//Name[1]')** returns the contents of the first `Name` element in the document. Note that the expression `//Name[1]` is passed as a string by enclosing it in single quotes.

The `altova:evaluate` function can optionally take additional arguments. These arguments are the values of in-scope variables that have the names `p1`, `p2`, `p3`... `pN`. Note the following points about usage: (i) The variables must be defined with names of the form `pX`, where `X` is an integer; (ii) the `altova:evaluate` function's arguments (*see signature above*), from the second argument onwards, provide the values of the variables, with the sequence of the arguments corresponding to the numerically ordered sequence of variables: `p1` to `pN`: The second argument will be the value of the variable `p1`, the third argument that of the variable `p2`, and so on; (iii) The variable values must be of type `item*`.

Example

```
<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
<xsl:value-of select="altova:evaluate($xpath, 10, 20, 'hi')" />
```

outputs `"hi 20 10"`

In the listing above, notice the following:

- The second argument of the `altova:evaluate` expression is the value assigned to the variable `$p1`, the third argument that assigned to the variable `$p2`, and so on.
- Notice that the fourth argument of the function is a string value, indicated by its being enclosed in quotes.

© 2015 Altova GmbH

- The `select` attribute of the `xs:variable` element supplies the XPath expression. Since this expression must be of type `xs:string`, it is enclosed in single quotes.

Examples to further illustrate the use of variables

- `<xsl:variable name="xpath" select="'$p1'" />`
  `<xsl:value-of select="altova:evaluate($xpath, //Name[1])" />`
  Outputs value of the first `Name` *element.*

- `<xsl:variable name="xpath" select="'$p1'" />`
  `<xsl:value-of select="altova:evaluate($xpath, '//Name[1]')" />`
  Outputs "`//Name[1]`*"*

The `altova:evaluate()` extension function is useful in situations where an XPath expression in the XSLT stylesheet contains one or more parts that must be evaluated dynamically. For example, consider a situation in which a user enters his request for the sorting criterion and this criterion is stored in the attribute `UserReq/@sortkey`. In the stylesheet, you could then have the expression: `<xsl:sort select="`**`altova:evaluate(../UserReq/@sortkey)`**`" order="ascending"/>`. The `altova:evaluate()` function reads the `sortkey` attribute of the `UserReq` child element of the parent of the context node. Say the value of the `sortkey` attribute is `Price`, then `Price` is returned by the `altova:evaluate()` function and becomes the value of the `select` attribute: `<xsl:sort select="`**`Price`**`" order="ascending"/>`. If this `sort` instruction occurs within the context of an element called `Order`, then the `Order` elements will be sorted according to the values of their `Price` children. Alternatively, if the value of `@sortkey` were, say, `Date`, then the `Order` elements would be sorted according to the values of their `Date` children. So the sort criterion for `Order` is selected from the `sortkey` attribute at runtime. This could not have been achieved with an expression like: `<xsl:sort select="`**`../UserReq/@sortkey`**`" order="ascending"/>`. In the case shown above, the sort criterion would be the `sortkey` attribute itself, not `Price` or `Date` (or any other current content of `sortkey`).

Note: The static context includes namespaces, types, and functions—but not variables—from the calling environment. The base URI and default namespace are inherited.

More examples

- Static variables: `<xsl:value-of select="$i3, $i2, $i1" />`
  Outputs the values of three variables.

- Dynamic XPath expression with dynamic variables:
  `<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />`
  `<xsl:value-of select="altova:evaluate($xpath, 10, 20, 30)" />`
  Outputs "`30 20 10`*"*

- Dynamic XPath expression with no dynamic variable:
  `<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />`
  `<xsl:value-of select="altova:evaluate($xpath)" />`
  Outputs error: No variable defined for $p3.

▼ **encode-for-rtf [altova:]**

altova:encode-for-rtf(input *as xs:string*, **preserveallwhitespace** *as xs:boolean*, **preservenewlines** *as xs:boolean*) as **xs:string** XSLT2 XSLT3
Converts the input string into code for RTF. Whitespace and new lines will be preserved according to the boolean value specified for their respective arguments.

[Top]

XBRL functions

Altova XBRL functions can be used only with editions of Altova products that have XBRL support.

▼ **xbrl-footnotes [altova:]**

altova:xbrl-footnotes(*node()***)** as **node()*** XSLT2 XSLT3
Takes a node as its input argument and returns the set of XBRL footnote nodes referenced by the input node.

▼ **xbrl-labels [altova:]**

altova:xbrl-labels(*xs:QName*, *xs:string***)** as **node()*** XSLT2 XSLT3
Takes two input arguments: a node name and the taxonomy file location containing the node. The function returns the XBRL label nodes associated with the input node.

[Top]

XPath/XQuery Functions: Date and Time

Altova's date/time extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data held as XML Schema's various date and time datatypes. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, http://www.altova.com/xslt-extensions**, and are indicated in this section with the prefix **altova:**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3

XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

▼ Grouped by functionality

- Add duration to xs:dateTime and return xs:dateTime
- Add a duration to xs:date and return xs:date
- Add a duration to xs:time and return xs:time
- Format and retrieve durations
- Remove timezone from functions that generate current date/time
- Return weekday as integer from date
- Return week number as integer from date
- Build date, time, or duration type from lexical components of each type
- Construct date, dateTime, or time type from string input
- Age-related functions

▼ Grouped alphabetically

altova:add-days-to-date
altova:add-days-to-dateTime
altova:add-hours-to-dateTime
altova:add-hours-to-time
altova:add-minutes-to-dateTime
altova:add-minutes-to-time
altova:add-months-to-date
altova:add-months-to-dateTime
altova:add-seconds-to-dateTime
altova:add-seconds-to-time
altova:add-years-to-date
altova:add-years-to-dateTime
altova:age
altova:age-details
altova:build-date
altova:build-duration
altova:build-time
altova:current-dateTime-no-TZ
altova:current-date-no-TZ
altova:current-time-no-TZ
altova:format-duration
altova:parse-date
altova:parse-dateTime
altova:parse-duration
altova:parse-time
altova:weekday-from-date
altova:weekday-from-dateTime
altova:weeknumber-from-date
altova:weeknumber-from-dateTime

[Top]

Add a duration to `xs:dateTime` XP3 XQ3

These functions add a duration to **`xs:dateTime`** and return **`xs:dateTime`**. The `xs:dateTime` type

has a format of `CCYY-MM-DDThh:mm:ss.sss`. This is a concatenation of the `xs:date` and `xs:time` formats separated by the letter `T`. A timezone suffix`+01:00 (for example)` is optional.

add-years-to-dateTime [altova:]

altova:add-years-to-dateTime(DateTime *as xs:dateTime*, **Years** *as xs:integer*) as **xs:dateTime** XP3 XQ3

Adds a duration in years to an `xs:dateTime` (*see examples below*). The second argument is the number of years to be added to the `xs:dateTime` supplied as the first argument. The result is of type `xs:dateTime`.

Examples

- **altova:add-years-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), 10) returns 2024-01-15T14:00:00
- **altova:add-years-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), -4) returns 2010-01-15T14:00:00

add-months-to-dateTime [altova:]

altova:add-months-to-dateTime(DateTime *as xs:dateTime*, **Months** *as xs:integer*) as **xs:dateTime** XP3 XQ3

Adds a duration in months to an `xs:dateTime` (*see examples below*). The second argument is the number of months to be added to the `xs:dateTime` supplied as the first argument. The result is of type `xs:dateTime`.

Examples

- **altova:add-months-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), 10) returns 2014-11-15T14:00:00
- **altova:add-months-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), -2) returns 2013-11-15T14:00:00

add-days-to-dateTime [altova:]

altova:add-days-to-dateTime(DateTime *as xs:dateTime*, **Days** *as xs:integer*) as **xs:dateTime** XP3 XQ3

Adds a duration in days to an `xs:dateTime` (*see examples below*). The second argument is the number of days to be added to the `xs:dateTime` supplied as the first argument. The result is of type `xs:dateTime`.

Examples

- **altova:add-days-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), 10) returns 2014-01-25T14:00:00
- **altova:add-days-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), -8) returns 2014-01-07T14:00:00

add-hours-to-dateTime [altova:]

altova:add-hours-to-dateTime(DateTime *as xs:dateTime*, **Hours** *as xs:integer*) as **xs:dateTime** XP3 XQ3

Adds a duration in hours to an `xs:dateTime` (*see examples below*). The second argument is the number of hours to be added to the `xs:dateTime` supplied as the first argument. The

© 2015 Altova GmbH

result is of type `xs:dateTime`.

Examples

- **altova:add-hours-to-dateTime**(xs:dateTime("2014-01-15T13:00:00"), 10) returns 2014-01-15T23:00:00
- **altova:add-hours-to-dateTime**(xs:dateTime("2014-01-15T13:00:00"), -8) returns 2014-01-15T05:00:00

add-minutes-to-dateTime [altova:]

altova:add-minutes-to-dateTime(DateTime *as xs:dateTime*, **Minutes** *as xs:integer***)** as **xs:dateTime** XP3 XQ3

Adds a duration in minutes to an `xs:dateTime` (*see examples below*). The second argument is the number of minutes to be added to the `xs:dateTime` supplied as the first argument. The result is of type `xs:dateTime`.

Examples

- **altova:add-minutes-to-dateTime**(xs:dateTime("2014-01-15T14:10:00"), 45) returns 2014-01-15T14:55:00
- **altova:add-minutes-to-dateTime**(xs:dateTime("2014-01-15T14:10:00"), -5) returns 2014-01-15T14:05:00

add-seconds-to-dateTime [altova:]

altova:add-seconds-to-dateTime(DateTime *as xs:dateTime*, **Seconds** *as xs:integer***)** as **xs:dateTime** XP3 XQ3

Adds a duration in seconds to an `xs:dateTime` (*see examples below*). The second argument is the number of seconds to be added to the `xs:dateTime` supplied as the first argument. The result is of type `xs:dateTime`.

Examples

- **altova:add-seconds-to-dateTime**(xs:dateTime("2014-01-15T14:00:10"), 20) returns 2014-01-15T14:00:30
- **altova:add-seconds-to-dateTime**(xs:dateTime("2014-01-15T14:00:10"), -5) returns 2014-01-15T14:00:05

[Top]

Add a duration to `xs:date` XP3 XQ3

These functions add a duration to **xs:date** and return **xs:date**. The `xs:date` type has a format of `CCYY-MM-DD`.

add-years-to-date [altova:]

altova:add-years-to-date(Date *as xs:date*, **Years** *as xs:integer***)** as **xs:date** XP3 XQ3

Adds a duration in years to a date. The second argument is the number of years to be added to the `xs:date` supplied as the first argument. The result is of type `xs:date`.

Examples

- **altova:add-years-to-date**(xs:date("2014-01-15"), 10) returns 2024-01-15
- **altova:add-years-to-date**(xs:date("2014-01-15"), -4) returns 2010-01-15

add-months-to-date [altova:]

altova:add-months-to-date(Date *as xs:date*, **Months** *as xs:integer***)** as **xs:date**
XP3 XQ3
Adds a duration in months to a date. The second argument is the number of months to be added to the xs:date supplied as the first argument. The result is of type xs:date.

Examples

- **altova:add-months-to-date**(xs:date("2014-01-15"), 10) returns 2014-11-15
- **altova:add-months-to-date**(xs:date("2014-01-15"), -2) returns 2013-11-15

add-days-to-date [altova:]

altova:add-days-to-date(Date *as xs:date*, **Days** *as xs:integer***)** as **xs:date** XP3 XQ3
Adds a duration in days to a date. The second argument is the number of days to be added to the xs:date supplied as the first argument. The result is of type xs:date.

Examples

- **altova:add-days-to-date**(xs:date("2014-01-15"), 10) returns 2014-01-25
- **altova:add-days-to-date**(xs:date("2014-01-15"), -8) returns 2014-01-07

[Top]

Format and retrieve durations XP3 XQ3

These functions add a duration to **xs:date** and return **xs:date**. The xs:date type has a format of CCYY-MM-DD.

format-duration [altova:]

altova:format-duration(Duration *as xs:duration*, **Picture** *as xs:string***)** as **xs:string** XP3 XQ3
Formats a duration, which is submitted as the first argument, according to a picture string submitted as the second argument. The output is a text string formatted according to the picture string.

Examples

- **altova:format-duration**(xs:duration("P2DT2H53M11.7S"), "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]") returns "Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7"
- **altova:format-duration**(xs:duration("P3M2DT2H53M11.7S"), "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]") returns "Months:03 Days:02 Hours:02 Minutes:53"

parse-duration [altova:]

 © 2015 Altova GmbH

altova:parse-duration(InputString *as xs:string*, **Picture** *as xs:string*) as **xs:duration** XP3 XQ3

Takes a patterned string as the first argument, and a picture string as the second argument. The input string is parsed on the basis of the picture string, and an `xs:duration` is returned.

Examples

- **altova:parse-duration**("Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7"), "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]") returns "P2DT2H53M11.7S"
- **altova:parse-duration**("Months:03 Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7", "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]") returns "P3M2DT2H53M"

[Top]

Add a duration to `xs:time` XP3 XQ3

These functions add a duration to **xs:time** and return **xs:time**. The `xs:time` type has a lexical form of `hh:mm:ss.sss`. An optional time zone may be suffixed. The letter `Z` indicates Coordinated Universal Time (UTC). All other time zones are represented by their difference from UTC in the format `+hh:mm`, or `-hh:mm`. If no time zone value is present, it is considered unknown; it is not assumed to be UTC.

add-hours-to-time [altova:]

altova:add-hours-to-time(Time *as xs:time*, **Hours** *as xs:integer*) as **xs:time** XP3 XQ3

Adds a duration in hours to a time. The second argument is the number of hours to be added to the `xs:time` supplied as the first argument. The result is of type `xs:time`.

Examples

- **altova:add-hours-to-time**(xs:time("11:00:00"), 10) returns 21:00:00
- **altova:add-hours-to-time**(xs:time("11:00:00"), -7) returns 04:00:00

add-minutes-to-time [altova:]

altova:add-minutes-to-time(Time *as xs:time*, **Minutes** *as xs:integer*) as **xs:time** XP3 XQ3

Adds a duration in minutes to a time. The second argument is the number of minutes to be added to the `xs:time` supplied as the first argument. The result is of type `xs:time`.

Examples

- **altova:add-minutes-to-time**(xs:time("14:10:00"), 45) returns 14:55:00
- **altova:add-minutes-to-time**(xs:time("14:10:00"), -5) returns 14:05:00

add-seconds-to-time [altova:]

altova:add-seconds-to-time(Time *as xs:time*, **Minutes** *as xs:integer*) as **xs:time** XP3 XQ3

Adds a duration in seconds to a time. The second argument is the number of seconds to be added to the `xs:time` supplied as the first argument. The result is of type `xs:time`. The

Seconds component can be in the range of `0` to `59.999`.

Examples

- **altova:add-seconds-to-time**(xs:time("14:00:00"), 20) returns 14:00:20
- **altova:add-seconds-to-time**(xs:time("14:00:00"), 20.895) returns 14:00:20.895

[Top]

Remove the timezone part from date/time datatypes XP3 XQ3

These functions remove the timezone from the current **xs:dateTime**, **xs:date**, or **xs:time** values, respectively. Note that the difference between `xs:dateTime` and `xs:dateTimeStamp` is that in the case of the latter the timezone part is required (while it is optional in the case of the former). So the format of an `xs:dateTimeStamp` value is: `CCYY-MM-DDThh:mm:ss.sss±hh:mm`. or `CCYY-MM-DDThh:mm:ss.sssZ`. If the date and time is read from the system clock as `xs:dateTimeStamp`, the `current-dateTime-no-TZ()` function can be used to remove the timezone if so required.

current-dateTime-no-TZ [altova:]

altova:current-dateTime-no-TZ() as **xs:dateTime** XP3 XQ3

This function takes no argument. It removes the timezone part of `current-dateTime()` (which is the current date-and-time according to the system clock) and returns an `xs:dateTime` value.

Examples

If the current dateTime is **2014-01-15T14:00:00+01:00**:

- **altova:current-dateTime-no-TZ**() returns 2014-01-15T14:00:00

current-date-no-TZ [altova:]

altova:current-date-no-TZ() as **xs:date** XP3 XQ3

This function takes no argument. It removes the timezone part of `current-date()` (which is the current date according to the system clock) and returns an `xs:date` value.

Examples

If the current date is **2014-01-15+01:00**:

- **altova:current-date-no-TZ**() returns 2014-01-15

current-time-no-TZ [altova:]

altova:current-time-no-TZ() as **xs:time** XP3 XQ3

This function takes no argument. It removes the timezone part of `current-time()` (which is the current time according to the system clock) and returns an `xs:time` value.

Examples

If the current time is **14:00:00+01:00**:

- **altova:current-time-no-TZ**() returns 14:00:00

 © 2015 Altova GmbH

[Top]

Return the weekday from xs:dateTime or xs:date XP3 XQ3

These functions return the weekday (as an integer) from xs:dateTime or xs:date. The days of the week are numbered (using the American format) from 1 to 7, with Sunday=1. In the European format, the week starts with Monday (=1). The American format, where Sunday=1, can be set by using the integer 0 where an integer is accepted to indicate the format.

weekday-from-dateTime [altova:]

altova:weekday-from-dateTime(DateTime *as xs:dateTime*) as **xs:integer** XP3 XQ3

Takes a date-with-time as its single argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with Sunday=1. If the European format is required (where Monday=1), use the other signature of this function (*see next signature below*).

Examples

- **altova:weekday-from-dateTime**(xs:dateTime("2014-02-03T09:00:00")) returns 2, which would indicate a Monday.

altova:weekday-from-dateTime(DateTime *as xs:dateTime*, **Format** *as xs:integer*) as **xs:integer** XP3 XQ3

Takes a date-with-time as its first argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with Monday=1. If the second (integer) argument is 0, then the weekdays are numbered 1 to 7 starting with Sunday=1. If the second argument is an integer other than 0, then Monday=1. If there is no second argument, the function is read as having the other signature of this function (*see previous signature*).

Examples

- **altova:weekday-from-dateTime**(xs:dateTime("2014-02-03T09:00:00"), 1) returns 1, which would indicate a Monday
- **altova:weekday-from-dateTime**(xs:dateTime("2014-02-03T09:00:00"), 4) returns 1, which would indicate a Monday
- **altova:weekday-from-dateTime**(xs:dateTime("2014-02-03T09:00:00"), 0) returns 2, which would indicate a Monday.

weekday-from-date [altova:]

altova:weekday-from-date(Date *as xs:date*) as **xs:integer** XP3 XQ3

Takes a date as its single argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with Sunday=1. If the European format is required (where Monday=1), use the other signature of this function (*see next signature below*).

Examples

- **altova:weekday-from-date**(xs:date("2014-02-03+01:00")) returns 2, which would indicate a Monday.

altova:weekday-from-date(Date *as xs:date*, **Format** *as xs:integer*) as **xs:integer** XP3 XQ3

Takes a date as its first argument and returns the day of the week of this date as an integer.

The weekdays are numbered starting with `Monday=1`. If the second (`Format`) argument is `0`, then the weekdays are numbered `1` to `7` starting with `Sunday=1`. If the second argument is an integer other than `0`, then `Monday=1`. If there is no second argument, the function is read as having the other signature of this function (*see previous signature*).

Examples

- **`altova:weekday-from-date`**`(xs:date("2014-02-03"), 1)` returns `1`, which would indicate a Monday
- **`altova:weekday-from-date`**`(xs:date("2014-02-03"), 4)` returns `1`, which would indicate a Monday
- **`altova:weekday-from-date`**`(xs:date("2014-02-03"), 0)` returns `2`, which would indicate a Monday.

[Top]

Return the week number from `xs:dateTime` or `xs:date` XP2 XQ1 XP3 XQ3

These functions return the week number (as an integer) from `xs:dateTime` or `xs:date`. Week-numbering is available in the US, ISO/European, and Islamic calendar formats. Week-numbering is different in these calendar formats because the week is considered to start on different days (on Sunday in the US format, Monday in the ISO/European format, and Saturday in the Islamic format).

weeknumber-from-date [altova:]

altova:weeknumber-from-date(Date *as xs:date*, **Calendar** *as xs:integer*) as **xs:integer** XP2 XQ1 XP3 XQ3

Returns the week number of the submitted **Date** argument as an integer. The second argument (**Calendar**) specifies the calendar system to follow.

Supported **Calendar** values are:

- **0 = US calendar** (*week starts Sunday*)
- **1 = ISO standard, European calendar** (*week starts Monday*)
- **2 = Islamic calendar** (*week starts Saturday*)

Default is **0**.

Examples

- **`altova:weeknumber-from-date`**`(xs:date("2014-03-23"), 0)` returns `13`
- **`altova:weeknumber-from-date`**`(xs:date("2014-03-23"), 1)` returns `12`
- **`altova:weeknumber-from-date`**`(xs:date("2014-03-23"), 2)` returns `13`
- **`altova:weeknumber-from-date`**`(xs:date("2014-03-23") )` returns `13`

The day of the date in the examples above (`2014-03-23`) is Sunday. So the US and Islamic calendars are one week ahead of the European calendar on this day.

weeknumber-from-dateTime [altova:]

altova:weeknumber-from-dateTime(DateTime *as xs:dateTime*, **Calendar** *as xs:integer*) as **xs:integer** XP2 XQ1 XP3 XQ3

Returns the week number of the submitted **DateTime** argument as an integer. The second argument (**Calendar**) specifies the calendar system to follow.
Supported **Calendar** values are:

- **0 = US calendar** (*week starts Sunday*)
- **1 = ISO standard, European calendar** (*week starts Monday*)
- **2 = Islamic calendar** (*week starts Saturday*)

Default is **0**.

Examples

- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00"), 0)` returns `13`
- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00"), 1)` returns `12`
- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00"), 2)` returns `13`
- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00") )` returns `13`

The day of the dateTime in the examples above (`2014-03-23T00:00:00`) is Sunday. So the US and Islamic calendars are one week ahead of the European calendar on this day.

[Top]

Build date, time, and duration datatypes from their lexical components XP3 XQ3

The functions take the lexical components of the `xs:date`, `xs:time`, or `xs:duration` datatype as input arguments and combine them to build the respective datatype.

build-date [altova:]

altova:build-date(Year *as xs:integer*, **Month** *as xs:integer*, **Date** *as xs:integer***)** as **xs:date** XP3 XQ3

The first, second, and third arguments are, respectively, the year, month, and date. They are combined to build a value of `xs:date` type. The values of the integers must be within the correct range of that particular date part. For example, the second argument (for the month part) should not be greater than `12`.

Examples

- `altova:build-date(2014, 2, 03)` returns `2014-02-03`

build-time [altova:]

altova:build-time(Hours *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer***)** as **xs:time** XP3 XQ3

The first, second, and third arguments are, respectively, the hour (`0` to `23`), minutes (`0` to `59`), and seconds (`0` to `59`) values. They are combined to build a value of `xs:time` type. The values of the integers must be within the correct range of that particular time part. For

example, the second (`Minutes`) argument should not be greater than `59`. To add a timezone part to the value, use the other signature of this function (*see next signature*).

Examples

- **altova:build-time**(23, 4, 57) returns 23:04:57

altova:build-time(Hours *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer*, **TimeZone** *as xs:string***)** as **xs:time** XP3 XQ3

The first, second, and third arguments are, respectively, the hour (`0` to `23`), minutes (`0` to `59`), and seconds (`0` to `59`) values. The fourth argument is a string that provides the timezone part of the value. The four arguments are combined to build a value of `xs:time` type. The values of the integers must be within the correct range of that particular time part. For example, the second (`Minutes`) argument should not be greater than `59`.

Examples

- **altova:build-time**(23, 4, 57, '+1') returns 23:04:57+01:00

build-duration [altova:]

altova:build-duration(Years *as xs:integer*, **Months** *as xs:integer***)** as **xs:yearMonthDuration** XP3 XQ3

Takes two arguments to build a value of type `xs:yearMonthDuration`. The first arguments provides the `Years` part of the duration value, while the second argument provides the `Months` part. If the second (`Months`) argument is greater than or equal to `12`, then the integer is divided by `12`; the quotient is added to the first argument to provide the `Years` part of the duration value while the remainder (of the division) provides the `Months` part. To build a duration of type `xs:dayTimeDuration`., see the next signature.

Examples

- **altova:build-duration**(2, 10) returns P2Y10M
- **altova:build-duration**(14, 27) returns P16Y3M
- **altova:build-duration**(2, 24) returns P4Y

altova:build-duration(Days *as xs:integer*, **Hours** *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer***)** as **xs:dayTimeDuration** XP3 XQ3

Takes four arguments and combines them to build a value of type `xs:dayTimeDuration`. The first argument provides the `Days` part of the duration value, the second, third, and fourth arguments provide, respectively, the `Hours`, `Minutes`, and `Seconds` parts of the duration value. Each of the three Time arguments is converted to an equivalent value in terms of the next higher unit and the result is used for calculation of the total duration value. For example, 72 seconds is converted to `1M+12S` (1 minute and 12 seconds), and this value is used for calculation of the total duration value. To build a duration of type `xs:yearMonthDuration`., see the previous signature.

Examples

- **altova:build-duration**(2, 10, 3, 56) returns P2DT10H3M56S
- **altova:build-duration**(1, 0, 100, 0) returns P1DT1H40M
- **altova:build-duration**(1, 0, 0, 3600) returns P1DT1H

[Top]

 © 2015 Altova GmbH

Construct date, dateTime, and time datatypes from string input XP2 XQ1 XP3 XQ3

These functions take strings as arguments and construct `xs:date`, `xs:dateTime`, or `xs:time` datatypes. The string is analyzed for components of the datatype based on a submitted pattern argument.

parse-date [altova:]

altova:parse-date(Date *as xs:string*, **DatePattern** *as xs:string***)** as **xs:date**
XP2 XQ1 XP3 XQ3

Returns the input string **Date** as an **xs:date** value. The second argument **DatePattern** specifies the pattern (sequence of components) of the input string. **DatePattern** is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

D	Date
M	Month
Y	Year

The pattern in **DatePattern** must match the pattern in **Date**. Since the output is of type **xs:date**, the output will always have the lexical format **YYYY-MM-DD**.

Examples

- **altova:parse-date**(xs:string("09-12-2014"), "[D]-[M]-[Y]") returns 2014-12-09
- **altova:parse-date**(xs:string("09-12-2014"), "[M]-[D]-[Y]") returns 2014-09-12
- **altova:parse-date**("06/03/2014", "[M]/[D]/[Y]") returns 2014-06-03
- **altova:parse-date**("06 03 2014", "[M] [D] [Y]") returns 2014-06-03
- **altova:parse-date**("6 3 2014", "[M] [D] [Y]") returns 2014-06-03

parse-dateTime [altova:]

altova:parse-dateTime(DateTime *as xs:string*, **DateTimePattern** *as xs:string***)** as **xs:dateTime** XP2 XQ1 XP3 XQ3

Returns the input string **DateTime** as an **xs:dateTime** value.The second argument **DateTimePattern** specifies the pattern (sequence of components) of the input string. **DateTimePattern** is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

D	Date
M	Month
Y	Year
H	Hour
m	minutes
s	seconds

The pattern in **DateTimePattern** must match the pattern in **DateTime**. Since the output is of type **xs:dateTime**, the output will always have the lexical format **YYYY-MM-DDTHH:mm:ss**.

Examples

- **altova:parse-dateTime**(xs:string("09-12-2014 13:56:24"), "[M]-[D]-[Y] [H]:[m]:[s]") returns 2014-09-12T13:56:24
- **altova:parse-dateTime**("time=13:56:24; date=09-12-2014", "time=[H]:[m]:[s]; date=[D]-[M]-[Y]") returns 2014-12-09T13:56:24

parse-time [altova:]

altova:parse-time(Time *as xs:string*, TimePattern *as xs:string***)** as **xs:time**
XP2 XQ1 XP3 XQ3

Returns the input string **Time** as an **xs:time** value.The second argument **TimePattern** specifies the pattern (sequence of components) of the input string. **TimePattern** is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

H	Hour
m	minutes
s	seconds

The pattern in **TimePattern** must match the pattern in **Time**. Since the output is of type **xs:time**, the output will always have the lexical format **HH:mm:ss**.

Examples

- **altova:parse-time**(xs:string("13:56:24"), "[H]:[m]:[s]") returns 13:56:24
- **altova:parse-time**("13-56-24", "[H]-[m]") returns 13:56:00
- **altova:parse-time**("time=13h56m24s", "time=[H]h[m]m[s]s") returns 13:56:24
- **altova:parse-time**("time=24s56m13h", "time=[s]s[m]m[H]h") returns 13:56:24

[Top]

Age-related functions XP3 XQ3

These functions return the age as calculated (i) between one input argument date and the current date, or (ii) between two input argument dates. The **altova:age** function returns the age in terms of years, the **altova:age-details** function returns the age as a sequence of three integers giving the years, months, and days of the age.

age [altova:]

altova:age(StartDate *as xs:date***)** as **xs:integer** XP3 XQ3

Returns an integer that is the age *in years* of some object, counting from a start-date submitted as the argument and ending with the current date (taken from the system clock). If the input argument is a date anything greater than or equal to one year in the future, the return value will be negative.

Examples

© 2015 Altova GmbH

If the current date is **2014-01-15**:

- **altova:age**(xs:date("2013-01-15")) returns 1
- **altova:age**(xs:date("2013-01-16")) returns 0
- **altova:age**(xs:date("2015-01-15")) returns -1
- **altova:age**(xs:date("2015-01-14")) returns 0

altova:age(StartDate *as xs:date*, **EndDate** *as xs:date***)** as **xs:integer** XP3 XQ3

Returns an integer that is the age *in years* of some object, counting from a start-date that is submitted as the first argument up to an end-date that is the second argument. The return value will be negative if the first argument is one year or more later than the second argument.

Examples

If the current date is **2014-01-15**:

- **altova:age**(xs:date("2000-01-15"), xs:date("2010-01-15")) returns 10
- **altova:age**(xs:date("2000-01-15"), current-date()) returns 14 if the current date is 2014-01-15
- **altova:age**(xs:date("2014-01-15"), xs:date("2010-01-15")) returns -4

age-details [altova:]

altova:age-details(InputDate *as xs:date***)** as (**xs:integer**)* XP3 XQ3

Returns three integers that are, respectively, the years, months, and days between the date that is submitted as the argument and the current date (taken from the system clock). The sum of the returned years+months+days together gives the total time difference between the two dates (the input date and the current date). The input date may have a value earlier or later than the current date, but whether the input date is earlier or later is not indicated by the sign of the return values; the return values are always positive.

Examples

If the current date is **2014-01-15**:

- **altova:age-details**(xs:date("2014-01-16")) returns (0 0 1)
- **altova:age-details**(xs:date("2014-01-14")) returns (0 0 1)
- **altova:age-details**(xs:date("2013-01-16")) returns (1 0 1)
- **altova:age-details**(current-date()) returns (0 0 0)

altova:age-details(Date-1 *as xs:date*, **Date-2** *as xs:date***)** as (**xs:integer**)* XP3 XQ3

Returns three integers that are, respectively, the years, months, and days between the two argument dates. The sum of the returned years+months+days together gives the total time difference between the two input dates; it does not matter whether the earlier or later of the two dates is submitted as the first argument. The return values do not indicate whether the input date occurs earlier or later than the current date. Return values are always positive.

Examples

- **altova:age-details**(xs:date("2014-01-16"), xs:date("2014-01-15")) returns (0 0 1)
- **altova:age-details**(xs:date("2014-01-15"), xs:date("2014-01-16")) returns (0 0 1)

[Top]

XPath/XQuery Functions: Geolocation

The following geolocation XPath/XQuery extension functions are supported in the current version of MapForce and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

parse-geolocation [altova:]

altova:parse-geolocation(GeolocationInputString *as xs:string***)** as **xs:decimal+**
XP3 XQ3

Parses the supplied `GeolocationInputString` argument and returns the geolocation's latitude and longitude (in that order) as a sequence two `xs:decimal` items. The formats in which the geolocation input string can be supplied are listed below.

Note: The `image-exif-data` function and the Exif metadata's `@Geolocation` attribute can be used to supply the geolocation input string (*see example below*).

Examples

- **`altova:parse-geolocation`**`("33.33  -22.22")` returns the sequence of two `xs:decimals (33.33, 22.22)`
- **`altova:parse-geolocation`**`("48°51'29.6""N  24°17'40.2""")` returns the sequence of two `xs:decimals (48.8582222222222, 24.2945)`
- **`altova:parse-geolocation`**`('48°51''29.6"N  24°17''40.2"')` returns the sequence of two `xs:decimals (48.8582222222222, 24.2945)`
- **`altova:parse-geolocation`**`( `**`image-exif-data`**`(//MyImages/Image20141130.01)`**`/@Geolocation`**` )` returns a sequence of two `xs:decimals`

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

 © 2015 Altova GmbH

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (**"**) while unit indicators that are escaped are highlighted in blue (**""**).

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `W/E`)
  `D°M'S.SS"N/S  D°M'S.SS"W/E`
  Example: **`33°55'11.11"N  22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/W`**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **`33°55'11.11"  -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `W/E`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/W`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `W/E`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/W`**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **`33.33  -22.22`**

Examples of format-combinations:

```
33.33N  -22°44'55.25"
33.33  22°44'55.25"W
33.33  22.45
```

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **`Geolocation`** from standard Exif metadata tags. **`Geolocation`** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

geolocation-distance-km [altova:]

altova:geolocation-distance-km(GeolocationInputString-1 *as xs:string*, GeolocationInputString-2 *as xs:string***)** as **xs:decimal** XP3 XQ3

Calculates the distance between two geolocations in kilometers. The formats in which the geolocation input string can be supplied are listed below. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The image-exif-data function and the Exif metadata's @Geolocation attribute can be used to supply geolocation input strings.

Examples

- **altova:geolocation-distance-km**("33.33 -22.22", "48°51'29.6""N 24° 17'40.2""") returns the xs:decimal 4183.08132372392

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (") while unit indicators that are escaped are highlighted in blue ("").

- Degrees, minutes, decimal seconds, with suffixed orientation (N/S, W/E)
  D°M'S.SS"N/S D°M'S.SS"W/E
  Example: **33°55'11.11"N 22°44'55.25"W**

- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/W) is optional
  +/-D°M'S.SS" +/-D°M'S.SS"
  Example: **33°55'11.11" -22°44'55.25"**

- Degrees, decimal minutes, with suffixed orientation (N/S, W/E)
  D°M.MM'N/S D°M.MM'W/E
  Example: **33°55.55'N 22°44.44'W**

- Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (N/W) is optional
  +/-D°M.MM' +/-D°M.MM'
  Example: **+33°55.55' -22°44.44'**

© 2015 Altova GmbH

- Decimal degrees, with suffixed orientation (N/S, W/E)
  D.DDN/S D.DDW/E
  Example: **33.33N 22.22W**

- Decimal degrees, with prefixed sign (+/-); the plus sign for (**N/W**) is optional
  +/-D.DD +/-D.DD
  Example: **33.33 -22.22**

Examples of format-combinations:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

geolocation-distance-mi [altova:]

altova:geolocation-distance-mi(GeolocationInputString-1 *as xs:string*, **GeolocationInputString-2** *as xs:string*) as **xs:decimal** XP3 XQ3
Calculates the distance between two geolocations in miles. The formats in which a geolocation input string can be supplied are listed below. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The image-exif-data function and the Exif metadata's @Geolocation attribute can be used to supply geolocation input strings.

Examples

- **altova:geolocation-distance-mi**("33.33 -22.22", "48°51'29.6""N 24° 17'40.2""") returns the xs:decimal 2599.40652340653

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (**''**) while unit indicators that are escaped are highlighted in blue (**''''**).

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `W/E`)
  `D°M'S.SS"N/S  D°M'S.SS"W/E`
  Example: **`33°55'11.11"N  22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/W`**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **`33°55'11.11"  -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `W/E`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/W`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `W/E`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/W`**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **`33.33  -22.22`**

Examples of format-combinations:

```
33.33N  -22°44'55.25"
33.33  22°44'55.25"W
33.33  22.45
```

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **`Geolocation`** from standard Exif metadata tags. **`Geolocation`** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

© 2015 Altova GmbH

geolocation-within-polygon [altova:]

altova:geolocation-within-polygon(Geolocation *as xs:string*, **((PolygonPoint** *as xs:string*)+)) as **xs:boolean** XP3 XQ3

Determines whether **Geolocation** (the first argument) is within the polygonal area described by the **PolygonPoint** arguments. If the `PolygonPoint` arguments do not form a closed figure (formed when the first point and the last point are the same), then the first point is implicitly added as the last point in order to close the figure. All the arguments (`Geolocation` and `PolygonPoint+`) are given by geolocation input strings (*formats listed below*). If the `Geolocation` argument is within the polygonal area, then the function returns `true()`; otherwise it returns `false()`. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: The `image-exif-data` function and the Exif metadata's `@Geolocation` attribute can be used to supply geolocation input strings.

Examples

- **altova:geolocation-within-polygon(**"33 -22", ("58 -32", "-78 -55", "48 24", "58 -32"))** returns `true()`
- **altova:geolocation-within-polygon(**"33 -22", ("58 -32", "-78 -55", "48 24"))** returns `true()`
- **altova:geolocation-within-polygon(**"33 -22", ("58 -32", "-78 -55", "48°51'29.6""N 24°17'40.2""")) returns `true()`

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (") while unit indicators that are escaped are highlighted in blue ("").

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `W/E`)
  D°M'S.SS"N/S D°M'S.SS"W/E
  Example: **33°55'11.11"N 22°44'55.25"W**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**N/W**) is optional
  +/-D°M'S.SS" +/-D°M'S.SS"
  Example: **33°55'11.11" -22°44'55.25"**

- Degrees, decimal minutes, with suffixed orientation (N/S, W/E)
  D°M.MM'N/S D°M.MM'W/E
  Example: **33°55.55'N 22°44.44'W**

- Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (**N/W**) is optional
  +/-D°M.MM' +/-D°M.MM'
  Example: **+33°55.55' -22°44.44'**

- Decimal degrees, with suffixed orientation (N/S, W/E)
  D.DDN/S D.DDW/E
  Example: **33.33N 22.22W**

- Decimal degrees, with prefixed sign (+/-); the plus sign for (**N/W**) is optional
  +/-D.DD +/-D.DD
  Example: **33.33 -22.22**

Examples of format-combinations:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

geolocation-within-rectangle [altova:]

altova:geolocation-within-rectangle(Geolocation *as xs:string*, **RectCorner-1** *as xs:string*, **RectCorner-2** *as xs:string***)** as **xs:boolean** XP3 XQ3
Determines whether **Geolocation** (the first argument) is within the rectangle defined by the second and third arguments, **RectCorner-1** and **RectCorner-2**, which specify opposite corners of the rectangle. All the arguments (Geolocation, **RectCorner-1** and **RectCorner-2**) are given by geolocation input strings (*formats listed below*). If the Geolocation argument is within the rectangle, then the function returns true(); otherwise it returns false(). Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The image-exif-data function and the Exif metadata's @Geolocation attribute can be used to supply geolocation input strings.

© 2015 Altova GmbH

Examples

- **altova:geolocation-within-rectangle(**"33 -22", "58 -32", "-48 24"**)** returns true()
- **altova:geolocation-within-rectangle(**"33 -22", "58 -32", "48 24"**)** returns false()
- **altova:geolocation-within-rectangle(**"33 -22", "58 -32", **"**48°51'29.6**""**S 24°17'40.2**"""**) returns true()

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (**"**) while unit indicators that are escaped are highlighted in blue (**""**).

- Degrees, minutes, decimal seconds, with suffixed orientation (N/S, W/E)
  D°M'S.SS"N/S D°M'S.SS"W/E
  Example: **33°55'11.11"N 22°44'55.25"W**

- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (**N/W**) is optional
  +/-D°M'S.SS" +/-D°M'S.SS"
  Example: **33°55'11.11" -22°44'55.25"**

- Degrees, decimal minutes, with suffixed orientation (N/S, W/E)
  D°M.MM'N/S D°M.MM'W/E
  Example: **33°55.55'N 22°44.44'W**

- Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (**N/W**) is optional
  +/-D°M.MM' +/-D°M.MM'
  Example: **+33°55.55' -22°44.44'**

- Decimal degrees, with suffixed orientation (N/S, W/E)
  D.DDN/S D.DDW/E
  Example: **33.33N 22.22W**

- Decimal degrees, with prefixed sign (+/-); the plus sign for (**N/W**) is optional
  +/-D.DD +/-D.DD
  Example: **33.33 -22.22**

Examples of format-combinations:
33.33N -22°44'55.25"

```
33.33  22°44'55.25"W
33.33  22.45
```

- *Altova Exif Attribute: Geolocation*

  The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

[Top]

XPath/XQuery Functions: Image-Related

The following image-related XPath/XQuery extension functions are supported in the current version of MapForce and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, http://www.altova.com/xslt-extensions**, and are indicated in this section with the prefix **altova:**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

suggested-image-file-extension [altova:]

altova:suggested-image-file-extension(Base64String *as string***)** as **string?** XP3 XQ3

Takes the Base64 encoding of an image file as its argument and returns the file extension of the image as recorded in the Base64-encoding of the image. The returned value is a suggestion based on the image type information available in the encoding. If this information

is not available, then an empty string is returned. This function is useful if you wish to save a Base64 image as a file and wish to dynamically retrieve an appropriate file extension.

Examples

- `altova:suggested-image-file-extension(/MyImages/MobilePhone/Image20141130.01)` returns `'jpg'`
- `altova:suggested-image-file-extension($XML1/Staff/Person/@photo)` returns `''`

In the examples above, the nodes supplied as the argument of the function are assumed to contain a Base64-encoded image. The first example retrieves `jpg` as the file's type and extension. In the second example, the submitted Base64 encoding does not provide usable file extension information.

image-exif-data [altova:]

altova:image-exif-data(Base64BinaryString *as string***)** as **element?** **XP3** **XQ3**

Takes a Base64-encoded image as its argument and returns an element called **Exif** that contains the Exif metadata of the image. The Exif metadata is created as attribute-value pairs of the `Exif` element. The attribute names are the Exif data tags found in the Base64 encoding. The list of Exif-specification tags is given below. If a vendor-specific tag is present in the Exif data, this tag and its value will also be returned as an attribute-value pair. Additional to the standard Exif metadata tags (*see list below*), Altova-specific attribute-value pairs are also generated. These Altova Exif attributes are listed below.

Examples

- To access any one attribute, use the function like this:
  `image-exif-data(//MyImages/Image20141130.01)/@GPSLatitude`
  `image-exif-data(//MyImages/Image20141130.01)/@Geolocation`
- To access all the attributes, use the function like this:
  `image-exif-data(//MyImages/Image20141130.01)/@*`
- To access the names of all the attributes, use the following expression:
  `for $i in image-exif-data(//MyImages/Image20141130.01)/@* return name($i)`
  This is useful to find out the names of the attributes returned by the function.

Altova Exif Attribute: Geolocation

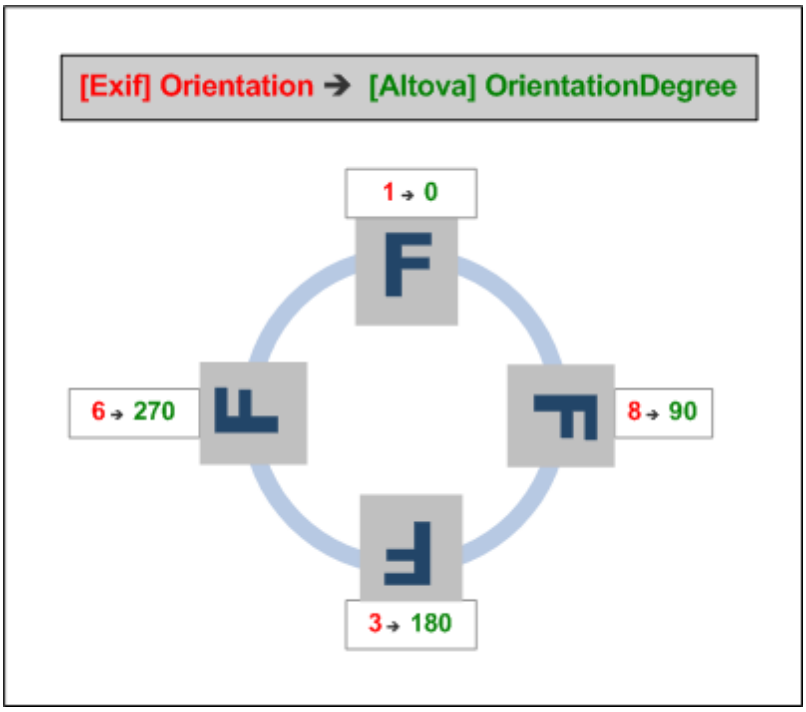
The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

Altova Exif Attribute: OrientationDegree

The Altova XPath/XQuery Engine generates the custom attribute **`OrientationDegree`** from the Exif metadata tag **`Orientation`**.

`OrientationDegree` translates the standard Exif tag `Orientation` from an integer value (`1`, `8`, `3`, or `6`) to the respective degree values of each (`0`, `90`, `180`, `270`), as shown in the figure below. Note that there are no translations of the `Orientation` values of `2`, `4`, `5`, `7`. (These orientations are obtained by flipping image `1` across its vertical center axis to get the image with a value of `2`, and then rotating this image in 90-degree jumps clockwise to get the values of `7`, `4`, and `5`, respectively).



Listing of standard Exif meta tags

- `ImageWidth`
- `ImageLength`
- `BitsPerSample`
- `Compression`
- `PhotometricInterpretation`
- `Orientation`
- `SamplesPerPixel`
- `PlanarConfiguration`
- `YCbCrSubSampling`
- `YCbCrPositioning`
- `XResolution`
- `YResolution`
- `ResolutionUnit`
- `StripOffsets`

© 2015 Altova GmbH

- RowsPerStrip
- StripByteCounts
- JPEGInterchangeFormat
- JPEGInterchangeFormatLength
- TransferFunction
- WhitePoint
- PrimaryChromaticities
- YCbCrCoefficients
- ReferenceBlackWhite
- DateTime
- ImageDescription
- Make
- Model
- Software
- Artist
- Copyright

- ExifVersion
- FlashpixVersion
- ColorSpace
- ComponentsConfiguration
- CompressedBitsPerPixel
- PixelXDimension
- PixelYDimension
- MakerNote
- UserComment
- RelatedSoundFile
- DateTimeOriginal
- DateTimeDigitized
- SubSecTime
- SubSecTimeOriginal
- SubSecTimeDigitized
- ExposureTime
- FNumber
- ExposureProgram
- SpectralSensitivity
- ISOSpeedRatings
- OECF
- ShutterSpeedValue
- ApertureValue
- BrightnessValue
- ExposureBiasValue
- MaxApertureValue
- SubjectDistance
- MeteringMode
- LightSource
- Flash
- FocalLength
- SubjectArea
- FlashEnergy
- SpatialFrequencyResponse
- FocalPlaneXResolution
- FocalPlaneYResolution
- FocalPlaneResolutionUnit
- SubjectLocation

- ExposureIndex
- SensingMethod
- FileSource
- SceneType
- CFAPattern
- CustomRendered
- ExposureMode
- WhiteBalance
- DigitalZoomRatio
- FocalLengthIn35mmFilm
- SceneCaptureType
- GainControl
- Contrast
- Saturation
- Sharpness
- DeviceSettingDescription
- SubjectDistanceRange
- ImageUniqueID

- GPSVersionID
- GPSLatitudeRef
- GPSLatitude
- GPSLongitudeRef
- GPSLongitude
- GPSAltitudeRef
- GPSAltitude
- GPSTimeStamp
- GPSSatellites
- GPSStatus
- GPSMeasureMode
- GPSDOP
- GPSSpeedRef
- GPSSpeed
- GPSTrackRef
- GPSTrack
- GPSImgDirectionRef
- GPSImgDirection
- GPSMapDatum
- GPSDestLatitudeRef
- GPSDestLatitude
- GPSDestLongitudeRef
- GPSDestLongitude
- GPSDestBearingRef
- GPSDestBearing
- GPSDestDistanceRef
- GPSDestDistance
- GPSProcessingMethod
- GPSAreaInformation
- GPSDateStamp
- GPSDifferential

[Top]

 © 2015 Altova GmbH

XPath/XQuery Functions: Numeric

Altova's numeric extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

Auto-numbering functions

generate-auto-number [altova:]

altova:generate-auto-number(ID *as xs:string*, **StartsWith** *as xs:double*, **Increment** *as xs:double*, **ResetOnChange** *as xs:string***)** as `xs:integer` XP1 XP2 XQ1 XP3 XQ3

Generates a number each time the function is called. The first number, which is generated the first time the function is called, is specified by the `StartsWith` argument. Each subsequent call to the function generates a new number, this number being incremented over the previously generated number by the value specified in the `Increment` argument. In effect, the `altova:generate-auto-number` function creates a counter having a name specified by the `ID` argument, with this counter being incremented each time the function is called. If the value of the `ResetOnChange` argument changes from that of the previous function call, then the value of the number to be generated is reset to the `StartsWith` value. Auto-numbering can also be reset by using the `altova:reset-auto-number` function.

Examples

- **`altova:generate-auto-number`**`("ChapterNumber", 1, 1, "SomeString")` will return one number each time the function is called, starting with `1`, and incrementing by `1` with each call to the function. As long as the fourth argument remains `"SomeString"` in each subsequent call, the incrementing will continue. When the value of the fourth argument changes, the counter (called `ChapterNumber`) will reset to 1. The value of `ChapterNumber` can also be reset by a call to the `altova:reset-auto-number` function, like this: `altova:reset-auto-number("ChapterNumber")`.

reset-auto-number [altova:]

altova:reset-auto-number(ID *as xs:string***)** XP1 XP2 XQ1 XP3 XQ3

This function resets the number of the auto-numbering counter named in the `ID` argument. The number is reset to the number specified by the `StartsWith` argument of the `altova:generate-auto-number` function that created the counter named in the `ID` argument.

Examples

- **altova:reset-auto-number**(`"ChapterNumber"`) resets the number of the auto-numbering counter named `ChapterNumber` that was created by the `altova:generate-auto-number` function. The number is reset to the value of the `StartsWith` argument of the `altova:generate-auto-number` function that created `ChapterNumber`.

[Top]

Numeric functions

hex-string-to-integer [altova:]

altova:hex-string-to-integer(HexString *as xs:string***)** as **xs:integer** XP3 XQ3

Takes a string argument that is the Base-16 equivalent of an integer in the decimal system (Base-10), and returns the decimal integer.

Examples

- **altova:hex-string-to-integer**(`'1'`) returns `1`
- **altova:hex-string-to-integer**(`'9'`) returns `9`
- **altova:hex-string-to-integer**(`'A'`) returns `10`
- **altova:hex-string-to-integer**(`'B'`) returns `11`
- **altova:hex-string-to-integer**(`'F'`) returns `15`
- **altova:hex-string-to-integer**(`'G'`) returns an error
- **altova:hex-string-to-integer**(`'10'`) returns `16`
- **altova:hex-string-to-integer**(`'01'`) returns `1`
- **altova:hex-string-to-integer**(`'20'`) returns `32`
- **altova:hex-string-to-integer**(`'21'`) returns `33`
- **altova:hex-string-to-integer**(`'5A'`) returns `90`
- **altova:hex-string-to-integer**(`'USA'`) returns an error

integer-to-hex-string [altova:]

altova:integer-to-hex-string(Integer *as xs:integer***)** as **xs:string** XP3 XQ3

Takes an integer argument and returns its Base-16 equivalent as a string.

Examples

- **altova:integer-to-hex-string**(`1`) returns `'1'`
- **altova:integer-to-hex-string**(`9`) returns `'9'`
- **altova:integer-to-hex-string**(`10`) returns `'A'`
- **altova:integer-to-hex-string**(`11`) returns `'B'`
- **altova:integer-to-hex-string**(`15`) returns `'F'`
- **altova:integer-to-hex-string**(`16`) returns `'10'`

 © 2015 Altova GmbH

- `altova:integer-to-hex-string(32)` returns `'20'`
- `altova:integer-to-hex-string(33)` returns `'21'`
- `altova:integer-to-hex-string(90)` returns `'5A'`

[Top]

Number-formatting functions

generate-auto-number [altova:]

`altova:generate-auto-number(ID as xs:string, StartsWith as xs:double, Increment as xs:double, ResetOnChange as xs:string)` as `xs:integer` XP1 XP2 XQ1 XP3 XQ3

Generates a number each time the function is called. The first number, which is generated the first time the function is called, is specified by the `StartsWith` argument. Each subsequent call to the function generates a new number, this number being incremented over the previously generated number by the value specified in the `Increment` argument. In effect, the `altova:generate-auto-number` function creates a counter having a name specified by the `ID` argument, with this counter being incremented each time the function is called. If the value of the `ResetOnChange` argument changes from that of the previous function call, then the value of the number to be generated is reset to the `StartsWith` value. Auto-numbering can also be reset by using the `altova:reset-auto-number` function.

Examples

- `altova:generate-auto-number("ChapterNumber", 1, 1, "SomeString")` will return one number each time the function is called, starting with `1`, and incrementing by `1` with each call to the function. As long as the fourth argument remains `"SomeString"` in each subsequent call, the incrementing will continue. When the value of the fourth argument changes, the counter (called `ChapterNumber`) will reset to 1. The value of `ChapterNumber` can also be reset by a call to the `altova:reset-auto-number` function, like this: `altova:reset-auto-number("ChapterNumber")`.

[Top]

XPath/XQuery Functions: Sequence

Altova's sequence extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information

about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

attributes [altova:]

altova:attributes(AttributeName *as xs:string*) as **attribute()*** XP3 XQ3
Returns all attributes that have a local name which is the same as the name supplied in the input argument, `AttributeName`. The search is case-sensitive and conducted along the `attribute::` axis. This means that the context node must be the parent element node.

Examples

- **altova:attributes**("MyAttribute") returns MyAttribute()*

altova:attributes(AttributeName *as xs:string*, **SearchOptions** *as xs:string*) as **attribute()*** XP3 XQ3
Returns all attributes that have a local name which is the same as the name supplied in the input argument, `AttributeName`. The search is case-sensitive and conducted along the `attribute::` axis. The context node must be the parent element node. The second argument is a string containing option flags. Available flags are:
r = switches to a regular-expression search; `AttributeName` must then be a regular-expression search string;
f = If this option is specified, then `AttributeName` provides a full match; otherwise `AttributeName` need only partially match an attribute name to return that attribute. For example: if **f** is not specified, then `MyAtt` will return `MyAttribute`;
i = switches to a case-insensitive search;
p = includes the namespace prefix in the search; `AttributeName` should then contain the namespace prefix, for example: `altova:MyAttribute`.
The flags can be written in any order. Invalid flags will generate errors. One or more flags can be omitted. The empty string is allowed, and will produce the same effect as the function having only one argument (*previous signature*). However, an empty sequence is not allowed as the second argument.

Examples

- **altova:attributes**("MyAttribute", "rfip") returns MyAttribute()*
- **altova:attributes**("MyAttribute", "pri") returns MyAttribute()*
- **altova:attributes**("MyAtt", "rip") returns MyAttribute()*
- **altova:attributes**("MyAttributes", "rfip") returns no match
- **altova:attributes**("MyAttribute", "") returns MyAttribute()*
- **altova:attributes**("MyAttribute", "Rip") returns an unrecognized-flag error.
- **altova:attributes**("MyAttribute",) returns a missing-second-argument error.

elements [altova:]

altova:elements(ElementName *as xs:string*) as **element()*** XP3 XQ3
Returns all elements that have a local name which is the same as the name supplied in the input argument, `ElementName`. The search is case-sensitive and conducted along the

child:: axis. The context node must be the parent node of the element/s being searched for.

Examples

- **altova:elements**("MyElement") returns MyElement()*

altova:elements(ElementName *as xs:string*, **SearchOptions** *as xs:string*) as **element()*** **XP3 XQ3**

Returns all elements that have a local name which is the same as the name supplied in the input argument, ElementName. The search is case-sensitive and conducted along the child:: axis. The context node must be the parent node of the element/s being searched for. The second argument is a string containing option flags. Available flags are:
r = switches to a regular-expression search; ElementName must then be a regular-expression search string;
f = If this option is specified, then ElementName provides a full match; otherwise ElementName need only partially match an element name to return that element. For example: if **f** is not specified, then MyElem will return MyElement;
i = switches to a case-insensitive search;
p = includes the namespace prefix in the search; ElementName should then contain the namespace prefix, for example: altova:MyElement.
The flags can be written in any order. Invalid flags will generate errors. One or more flags can be omitted. The empty string is allowed, and will produce the same effect as the function having only one argument (*previous signature*). However, an empty sequence is not allowed.

Examples

- **altova:elements**("MyElement", "rip") returns MyElement()*
- **altova:elements**("MyElement", "pri") returns MyElement()*
- **altova:elements**("MyElement", "") returns MyElement()*
- **altova:attributes**("MyElem", "rip") returns MyElement()*
- **altova:attributes**("MyElements", "rfip") returns no match
- **altova:elements**("MyElement", "Rip") returns an unrecognized-flag error.
- **altova:elements**("MyElement",) returns a missing-second-argument error.

find-first [altova:]

altova:find-first((Sequence *as item()**), **(Condition(Sequence-Item** *as xs:boolean*)) as **item()?** **XP3 XQ3**

This function takes two arguments. The first argument is a sequence of one or more items of any datatype. The second argument, Condition, is a reference to an XPath function that takes one argument (has an arity of **1**) and returns a boolean. Each item of **Sequence** is submitted, in turn, to the function referenced in Condition. (*Remember:* This function takes a single argument.) The first **Sequence** item that causes the function in **Condition** to evaluate to **true()** is returned as the result of **altova:find-first**, and the iteration stops.

Examples

- **altova:find-first**(5 to 10, function($a) {$a mod 2 = 0}) returns xs:integer 6

  The **Condition** argument references the XPath 3.0 inline function, **function()**, which declares an inline function named **$a** and then defines it. Each item in the **Sequence** argument of **altova:find-first** is passed, in turn, to **$a** as its input value. The input

value is tested on the condition in the function definition (`$a mod 2 = 0`). The first input value to satisfy this condition is returned as the result of **`altova:find-first`** (in this case **6**).

- **`altova:find-first`**`((1 to 10), (function($a) {$a+3=7}))` returns `xs:integer 4`

Further examples

If the file **`C:\Temp\Customers.xml`** exists:

- **`altova:find-first`**`( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) )` returns `xs:string C:\Temp\Customers.xml`

If the file **`C:\Temp\Customers.xml`** does not exist, and **`http://www.altova.com/index.html`** exists:

- **`altova:find-first`**`( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) )` returns `xs:string http://www.altova.com/index.html`

If the file **`C:\Temp\Customers.xml`** does not exist, and **`http://www.altova.com/index.html`** also does not exist:

- **`altova:find-first`**`( ("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1) )` returns no result

Notes about the examples given above

- The XPath 3.0 function, `doc-available`, takes a single string argument, which is used as a URI, and returns `true` if a document node is found at the submitted URI. (The document at the submitted URI must therefore be an XML document.)
- The `doc-available` function can be used for **`Condition`**, the second argument of `altova:find-first`, because it takes only one argument (arity=1), because it takes an `item()` as input (a string which is used as a URI), and returns a boolean value.
- Notice that the `doc-available` function is only referenced, not called. The `#1` suffix that is attached to it indicates a function with an arity of 1. In its entirety `doc-available#1` simply means: *Use the doc-availabe() function that has arity=1, passing to it as its single argument, in turn, each of the items in the first sequence*. As a result, each of the two strings will be passed to **`doc-available()`**, which uses the string as a URI and tests whether a document node exists at the URI. If one does, the **`doc-available()`** evaluates to `true()` and that string is returned as the result of the **`altova:find-first`** function. *Note about the doc-available() function: Relative paths are resolved relative to the the current base URI, which is by default the URI of the XML document from which the function is loaded.*

`find-first-combination [altova:]`

`altova:find-first-combination((Seq-01` *`as item()*`***`), (Seq-02`** *`as item()*`***`), (Condition( Seq-01-Item, Seq-02-Item`** *`as xs:boolean`***`))`** `as` **`item()*`** **`XP3`** **`XQ3`**

This function takes three arguments:

- The first two arguments, **Seq-01** and **Seq-02**, are sequences of one or more items of any datatype.
- The third argument, **Condition**, is a reference to an XPath function that takes two arguments (has an arity of **2**) and returns a boolean.

The items of **Seq-01** and **Seq-02** are passed in ordered pairs (one item from each sequence making up a pair) as the arguments of the function in **Condition**. The pairs are ordered as follows.

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X1 Y2), (X1 Y3) ... (X1 Yn), (X2 Y1), (X2 Y2) ... (Xn Yn)
```

The first ordered pair that causes the **Condition** function to evaluate to **true()** is returned as the result of **altova:find-first-combination**. Note that: (i) If the **Condition** function iterates through the submitted argument pairs and does not once evaluate to **true()**, then **altova:find-first-combination** returns *No results*; (ii) The result of **altova:find-first-combination** will always be a pair of items (of any datatype) or no item at all.

Examples

- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a +$b = 32})` returns the sequence of `xs:integers (11, 21)`
- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a +$b = 33})` returns the sequence of `xs:integers (11, 22)`
- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a +$b = 34})` returns the sequence of `xs:integers (11, 23)`

find-first-pair [altova:]

`altova:find-first-pair((Seq-01 as item()*), (Seq-02 as item()*), (Condition( Seq-01-Item, Seq-02-Item as xs:boolean)) as item()*` XP3 XQ3

This function takes three arguments:

- The first two arguments, **Seq-01** and **Seq-02**, are sequences of one or more items of any datatype.
- The third argument, **Condition**, is a reference to an XPath function that takes two arguments (has an arity of **2**) and returns a boolean.

The items of **Seq-01** and **Seq-02** are passed in ordered pairs as the arguments of the function in **Condition**. The pairs are ordered as follows.

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)
```

The first ordered pair that causes the **Condition** function to evaluate to **true()** is returned as the result of **altova:find-first-pair**. Note that: (i) If the **Condition** function iterates through the submitted argument pairs and does not once evaluate to **true()**, then **altova:find-first-pair** returns *No results*; (ii) The result of **altova:find-first-pair** will always be a pair of items (of any datatype) or no item at all.

Examples

- **altova:find-first-pair**(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32}) returns the sequence of xs:integers (11, 21)
- **altova:find-first-pair**(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33}) returns *No results*

Notice from the two examples above that the ordering of the pairs is: (11, 21) (12, 22) (13, 23)...(20, 30). This is why the second example returns *No results* (because no ordered pair gives a sum of 33).

find-first-pair-pos [altova:]

altova:find-first-pair-pos((Seq-01 *as item()**)**, (Seq-02** *as item()**)**, (Condition(Seq-01-Item, Seq-02-Item** *as xs:boolean***))** as **xs:integer** **XP3** **XQ3**
This function takes three arguments:

- The first two arguments, **Seq-01** and **Seq-02**, are sequences of one or more items of any datatype.
- The third argument, **Condition**, is a reference to an XPath function that takes two arguments (has an arity of **2**) and returns a boolean.

The items of **Seq-01** and **Seq-02** are passed in ordered pairs as the arguments of the function in **Condition**. The pairs are ordered as follows.

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)
```

The index position of the first ordered pair that causes the **Condition** function to evaluate to **true()** is returned as the result of **altova:find-first-pair-pos**. Note that if the **Condition** function iterates through the submitted argument pairs and does not once evaluate to **true()**, then **altova:find-first-pair-pos** returns *No results*.

Examples

- **altova:find-first-pair-pos**(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32}) returns 1
- **altova:find-first-pair-pos**(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33}) returns *No results*

Notice from the two examples above that the ordering of the pairs is: (11, 21) (12, 22) (13, 23)...(20, 30). In the first example, the first pair causes the **Condition** function to evaluate to **true()**, and so its index position in the sequence, **1**, is returned. The second example returns *No results* because no pair gives a sum of 33.

find-first-pos [altova:]

altova:find-first-pos((Sequence *as item()**)**, (Condition(Sequence-Item** *as xs:boolean***))** as **xs:integer** **XP3** **XQ3**
This function takes two arguments. The first argument is a sequence of one or more items of any datatype. The second argument, Condition, is a reference to an XPath function that takes one argument (has an arity of **1**) and returns a boolean. Each item of **Sequence** is submitted, in turn, to the function referenced in Condition. (*Remember:* This function takes

a single argument.) The first **Sequence** item that causes the function in **Condition** to evaluate to **true()** has its index position in **Sequence** returned as the result of **altova:find-first-pos**, and the iteration stops.

Examples

- **altova:find-first-pos**(5 to 10, function($a) {$a mod 2 = 0}) returns xs:integer 2

The **Condition** argument references the XPath 3.0 inline function, **function()**, which declares an inline function named **$a** and then defines it. Each item in the **Sequence** argument of **altova:find-first-pos** is passed, in turn, to **$a** as its input value. The input value is tested on the condition in the function definition ($a mod 2 = 0). The index position in the sequence of the first input value to satisfy this condition is returned as the result of **altova:find-first-pos** (in this case **2**, since **6**, the first value (in the sequence) to satisfy the condition, is at index position **2** in the sequence).

- **altova:find-first-pos**((2 to 10), (function($a) {$a+3=7})) returns xs:integer 3

Further examples

If the file **C:\Temp\Customers.xml** exists:

- **altova:find-first-pos**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) returns 1

If the file **C:\Temp\Customers.xml** does not exist, and **http://www.altova.com/index.html** exists:

- **altova:find-first-pos**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) returns 2

If the file **C:\Temp\Customers.xml** does not exist, and **http://www.altova.com/index.html** also does not exist:

- **altova:find-first-pos**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) returns no result

Notes about the examples given above

- The XPath 3.0 function, doc-available, takes a single string argument, which is used as a URI, and returns true if a document node is found at the submitted URI. (The document at the submitted URI must therefore be an XML document.)
- The doc-available function can be used for **Condition**, the second argument of altova:find-first-pos, because it takes only one argument (arity=1), because it takes an item() as input (a string which is used as a URI), and returns a boolean value.
- Notice that the doc-available function is only referenced, not called. The #1 suffix that is attached to it indicates a function with an arity of 1. In its entirety doc-available#1 simply means: *Use the doc-availabe() function that has arity=1, passing to it as its single argument, in turn, each of the items in the first sequence*. As a result, each of the two strings will be passed to **doc-available()**, which uses the string as a URI and tests whether a document node exists at the URI. If one does, the **doc-available()** function evaluates to true() and the index position of that string in the sequence is returned as the result of the **altova:find-**

first-pos function. *Note about the doc-available() function: Relative paths are resolved relative to the the current base URI, which is by default the URI of the XML document from which the function is loaded.*

substitute-empty [altova:]

altova:substitute-empty(FirstSequence *as item()**, **SecondSequence** as item()***)** as **item()*** **XP3** **XQ3**

If `FirstSequence` is empty, returns `SecondSequence`. If `FirstSequence` is not empty, returns `FirstSequence`.

Examples

- **altova:substitute-empty**((1,2,3), (4,5,6)) returns (1,2,3)
- **altova:substitute-empty**((), (4,5,6)) returns (4,5,6)

XPath/XQuery Functions: String

Altova's string extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, http://www.altova.com/xslt-extensions**, and are indicated in this section with the prefix **altova:**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	**XP1** **XP2** **XP3**
XSLT functions (used in XPath expressions in XSLT):	**XSLT1** **XSLT2** **XSLT3**
XQuery functions (used in XQuery expressions in XQuery):	**XQ1** **XQ3**

camel-case [altova:]

altova:camel-case(InputString *as xs:string***)** as **xs:string** **XP3** **XQ3**

Returns the input string **InputString** in CamelCase. The string is analyzed using the regular expression **'\s'** (which is a shortcut for the whitespace character). The first non-whitespace character after a whitespace or sequence of consecutive whitespaces is capitalized. The first character in the output string is capitalized.

Examples

 © 2015 Altova GmbH

- **altova:camel-case**("max") returns Max
- **altova:camel-case**("max max") returns Max Max
- **altova:camel-case**("file01.xml") returns File01.xml
- **altova:camel-case**("file01.xml file02.xml") returns File01.xml File02.xml
- **altova:camel-case**("file01.xml file02.xml") returns File01.xml File02.xml
- **altova:camel-case**("file01.xml -file02.xml") returns File01.xml -file02.xml

altova:camel-case(InputString *as xs:string*, **SplitChars** *as xs:string*, **IsRegex** *as xs:boolean***)** as **xs:string** XP3 XQ3

Converts the input string **InputString** to camel case by using **SplitChars** to determine the character/s that trigger the next capitalization. **SplitChars** is used as a regular expression when **IsRegex = true()**, or as plain characters when **IsRegex = false()**. The first character in the output string is capitalized.

Examples

- **altova:camel-case**("setname getname", "set|get", true()) returns setName getName
- **altova:camel-case**("altova\documents\testcases", "\", false()) returns Altova\Documents\Testcases

char [altova:]

altova:char(Position *as xs:integer***)** as **xs:string** XP3 XQ3

Returns a string containing the character at the position specified by the Position argument, in the string obtained by converting the value of the context item to xs:string. The result string will be empty if no character exists at the index submitted by the Position argument.

Examples

If the context item is **1234ABCD**:

- **altova:char**(2) returns 2
- **altova:char**(5) returns A
- **altova:char**(9) returns the empty string.
- **altova:char**(-2) returns the empty string.

altova:char(InputString *as xs:string*, **Position** *as xs:integer***)** as **xs:string** XP3 XQ3

Returns a string containing the character at the position specified by the Position argument, in the string submitted as the InputString argument. The result string will be empty if no character exists at the index submitted by the Position argument.

Examples

- **altova:char**("2014-01-15", 5) returns -
- **altova:char**("USA", 1) returns U
- **altova:char**("USA", 10) returns the empty string.
- **altova:char**("USA", -2) returns the empty string.

first-chars [altova:]

altova:first-chars(X-Number *as xs:integer***)** as **xs:string** XP3 XQ3
Returns a string containing the first `X-Number` of characters of the string obtained by converting the value of the context item to `xs:string`.

Examples

If the context item is **1234ABCD**:

- **altova:first-chars**(2) returns 12
- **altova:first-chars**(5) returns 1234A
- **altova:first-chars**(9) returns 1234ABCD

altova:first-chars(InputString *as xs:string*, **X-Number** *as xs:integer***)** as **xs:string** XP3 XQ3
Returns a string containing the first `X-Number` of characters of the string submitted as the `InputString` argument.

Examples

- **altova:first-chars**("2014-01-15", 5) returns 2014-
- **altova:first-chars**("USA", 1) returns U

last-chars [altova:]

altova:last-chars(X-Number *as xs:integer***)** as **xs:string** XP3 XQ3
Returns a string containing the last `X-Number` of characters of the string obtained by converting the value of the context item to `xs:string`.

Examples

If the context item is **1234ABCD**:

- **altova:last-chars**(2) returns CD
- **altova:last-chars**(5) returns 4ABCD
- **altova:last-chars**(9) returns 1234ABCD

altova:last-chars(InputString *as xs:string*, **X-Number** *as xs:integer***)** as **xs:string** XP3 XQ3
Returns a string containing the last `X-Number` of characters of the string submitted as the `InputString` argument.

Examples

- **altova:last-chars**("2014-01-15", 5) returns 01-15
- **altova:last-chars**("USA", 10) returns USA

pad-string-left [altova:]

altova:pad-string-left(StringToPad *as xs:string*, **StringLength** *as xs:integer*, **PadCharacter** *as xs:string***)** as **xs:string** XP3 XQ3
The `PadCharacter` argument is a single character. It is padded to the left of the string to increase the number of characters in `StringToPad` so that this number equals the integer value of the `StringLength` argument. The `StringLength` argument can have any integer value (positive or negative), but padding will occur only if the value of `StringLength` is greater than the number of characters in `StringToPad`. If `StringToPad`. has more characters than the value of `StringLength`, then `StringToPad` is left unchanged.

Examples

© 2015 Altova GmbH

- **altova:pad-string-left**('AP', 1, 'Z') returns 'AP'
- **altova:pad-string-left**('AP', 2, 'Z') returns 'AP'
- **altova:pad-string-left**('AP', 3, 'Z') returns 'ZAP'
- **altova:pad-string-left**('AP', 4, 'Z') returns 'ZZAP'
- **altova:pad-string-left**('AP', -3, 'Z') returns 'AP'
- **altova:pad-string-left**('AP', 3, 'YZ') returns a pad-character-too-long error

pad-string-right [altova:]

altova:pad-string-right(StringToPad *as xs:string*, **StringLength** *as xs:integer*, **PadCharacter** *as xs:string***)** as **xs:string** XP3 XQ3

The PadCharacter argument is a single character. It is padded to the right of the string to increase the number of characters in StringToPad so that this number equals the integer value of the StringLength argument. The StringLength argument can have any integer value (positive or negative), but padding will occur only if the value of StringLength is greater than the number of characters in StringToPad. If StringToPad has more characters than the value of StringLength, then StringToPad is left unchanged.

Examples

- **altova:pad-string-right**('AP', 1, 'Z') returns 'AP'
- **altova:pad-string-right**('AP', 2, 'Z') returns 'AP'
- **altova:pad-string-right**('AP', 3, 'Z') returns 'APZ'
- **altova:pad-string-right**('AP', 4, 'Z') returns 'APZZ'
- **altova:pad-string-right**('AP', -3, 'Z') returns 'AP'
- **altova:pad-string-right**('AP', 3, 'YZ') returns a pad-character-too-long error

repeat-string [altova:]

altova:repeat-string(InputString *as xs:string*, **Repeats** *as xs:integer***)** as **xs:string** XP2 XQ1 XP3 XQ3

Generates a string that is composed of the first InputString argument repeated Repeats number of times.

Examples

- **altova:repeat-string**("Altova #", 3) returns "Altova #Altova #Altova #"

substring-after-last [altova:]

altova:substring-after-last(MainString *as xs:string*, **CheckString** *as xs:string***)** as **xs:string** XP3 XQ3

If CheckString is found in MainString, then the substring that occurs after CheckString in MainString is returned. If CheckString is not found in MainString, then the empty string is returned. If CheckString is an empty string, then MainString is returned in its entirety. If there is more than one occurrence of CheckString in MainString, then the substring after the last occurrence of CheckString is returned.

Examples

- **altova:substring-after-last**('ABCDEFGH', 'B') returns 'CDEFGH'
- **altova:substring-after-last**('ABCDEFGH', 'BC') returns 'DEFGH'

- **altova:substring-after-last**('ABCDEFGH', 'BD') returns ''
- **altova:substring-after-last**('ABCDEFGH', 'Z') returns ''
- **altova:substring-after-last**('ABCDEFGH', '') returns 'ABCDEFGH'
- **altova:substring-after-last**('ABCD-ABCD', 'B') returns 'CD'
- **altova:substring-after-last**('ABCD-ABCD-ABCD', 'BCD') returns ''

substring-before-last [altova:]

altova:substring-before-last(MainString *as xs:string*, **CheckString** *as xs:string*) as **xs:string** XP3 XQ3

If CheckString is found in MainString, then the substring that occurs before CheckString in MainString is returned. If CheckString is not found in MainString, or if CheckString is an empty string, then the empty string is returned. If there is more than one occurrence of CheckString in MainString, then the substring before the last occurrence of CheckString is returned.

Examples

- **altova:substring-before-last**('ABCDEFGH', 'B') returns 'A'
- **altova:substring-before-last**('ABCDEFGH', 'BC') returns 'A'
- **altova:substring-before-last**('ABCDEFGH', 'BD') returns ''
- **altova:substring-before-last**('ABCDEFGH', 'Z') returns ''
- **altova:substring-before-last**('ABCDEFGH', '') returns ''
- **altova:substring-before-last**('ABCD-ABCD', 'B') returns 'ABCD-A'
- **altova:substring-before-last**('ABCD-ABCD-ABCD', 'ABCD') returns 'ABCD-ABCD-'

substring-pos [altova:]

altova:substring-pos(StringToCheck *as xs:string*, **StringToFind** *as xs:string*) as **xs:integer** XP3 XQ3

Returns the character position of the first occurrence of StringToFind in the string StringToCheck. The character position is returned as an integer. The first character of StringToCheck has the position 1. If StringToFind does not occur within StringToCheck, the integer 0 is returned. To check for the second or a later occurrence of StringToCheck, use the next signature of this function.

Examples

- **altova:substring-pos**('Altova', 'to') returns 3
- **altova:substring-pos**('Altova', 'tov') returns 3
- **altova:substring-pos**('Altova', 'tv') returns 0
- **altova:substring-pos**('AltovaAltova', 'to') returns 3

altova:substring-pos(StringToCheck *as xs:string*, **StringToFind** *as xs:string*, **Integer** *as xs:integer*) as **xs:integer** XP3 XQ3

Returns the character position of StringToFind in the string, StringToCheck. The search for StringToFind starts from the character position given by the Integer argument; the character substring before this position is not searched. The returned integer, however, is the position of the found string within the *entire* string, StringToCheck. This signature is useful for finding the second or a later position of a string that occurs multiple times with the StringToCheck. If StringToFind does not occur within StringToCheck, the integer 0 is

returned.

Examples

- `altova:substring-pos('Altova', 'to', 1)` returns `3`
- `altova:substring-pos('Altova', 'to', 3)` returns `3`
- `altova:substring-pos('Altova', 'to', 4)` returns `0`
- `altova:substring-pos('Altova-Altova', 'to', 0)` returns `3`
- `altova:substring-pos('Altova-Altova', 'to', 4)` returns `10`

trim-string [altova:]

altova:trim-string(InputString *as xs:string*) as **xs:string** XP3 XQ3

This function takes an `xs:string` argument, removes any leading and trailing whitespace, and returns a "trimmed" `xs:string`.

Examples

- `altova:trim-string("  Hello World  ")` returns `"Hello World"`
- `altova:trim-string("Hello World  ")` returns `"Hello World"`
- `altova:trim-string("  Hello World")` returns `"Hello World"`
- `altova:trim-string("Hello World")` returns `"Hello World"`
- `altova:trim-string("Hello  World")` returns `"Hello  World"`

trim-string-left [altova:]

altova:trim-string-left(InputString *as xs:string*) as **xs:string** XP3 XQ3

This function takes an `xs:string` argument, removes any leading whitespace, and returns a left-trimmed `xs:string`.

Examples

- `altova:trim-string-left("  Hello World  ")` returns `"Hello World  "`
- `altova:trim-string-left("Hello World  ")` returns `"Hello World  "`
- `altova:trim-string-left("  Hello World")` returns `"Hello World"`
- `altova:trim-string-left("Hello World")` returns `"Hello World"`
- `altova:trim-string-left("Hello  World")` returns `"Hello  World"`

trim-string-right [altova:]

altova:trim-string-right(InputString *as xs:string*) as **xs:string** XP3 XQ3

This function takes an `xs:string` argument, removes any trailing whitespace, and returns a right-trimmed `xs:string`.

Examples

- `altova:trim-string-right("  Hello World  ")` returns `"  Hello World"`
- `altova:trim-string-right("Hello World  ")` returns `"Hello World"`
- `altova:trim-string-right("  Hello World")` returns `"  Hello World"`
- `altova:trim-string-right("Hello World")` returns `"Hello World"`
- `altova:trim-string-right("Hello  World")` returns `"Hello  World"`

XPath/XQuery Functions: Miscellaneous

The following general purpose XPath/XQuery extension functions are supported in the current version of MapForce and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT functions. Altova extension functions are in the **Altova extension functions namespace, `http://www.altova.com/xslt-extensions`**, and are indicated in this section with the prefix **`altova:`**, which is assumed to be bound to this namespace. Note that, in future versions of your product, support for a function might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

URI functions

▼ **`get-temp-folder [altova:]`**

`altova:get-temp-folder()` as `xs:string` XP2 XQ1 XP3 XQ3

This function takes no argument. It returns the path to the temporary folder of the current user.

⊟ *Examples*

- **`altova:get-temp-folder`**`()` would return, on a Windows machine, something like `C:\Users\<UserName>\AppData\Local\Temp\` as an `xs:string`.

[Top]

11.1.2.2 Miscellaneous Extension Functions

There are several ready-made functions in programming languages such as Java and C# that are not available as XQuery/XPath functions or as XSLT functions. A good example would be the math functions available in Java, such as `sin()` and `cos()`. If these functions were available to the designers of XSLT stylesheets and XQuery queries, it would increase the application area of stylesheets and queries and greatly simplify the tasks of stylesheet creators. The XSLT and XQuery engines used in a number of Altova products support the use of extension functions in Java and .NET, as well as MSXSL scripts for XSLT. This section describes how to use extension functions and MSXSL scripts in your XSLT stylesheets and XQuery documents. The available

 © 2015 Altova GmbH

extension functions are organized into the following sections:

- Java Extension Functions
- .NET Extension Functions
- MSXSL Scripts for XSLT

The two main issues considered in the descriptions are: (i) how functions in the respective libraries are called; and (ii) what rules are followed for converting arguments in a function call to the required input format of the function, and what rules are followed for the return conversion (function result to XSLT/XQuery data object).

Requirements

For extension functions support, a Java Runtime Environment (for access to Java functions) and .NET Framework 2.0 (minimum, for access to .NET functions) must be installed on the machine running the XSLT transformation or XQuery execution, or must be accessible for the transformations.

Java Extension Functions

A Java extension function can be used within an XPath or XQuery expression to invoke a Java constructor or call a Java method (static or instance).

A field in a Java class is considered to be a method without any argument. A field can be static or instance. How to access fields is described in the respective sub-sections, static and instance.

This section is organized into the following sub-sections:

- Java: Constructors
- Java: Static Methods and Static Fields
- Java: Instance Methods and Instance Fields
- Datatypes: XPath/XQuery to Java
- Datatypes: Java to XPath/XQuery

Form of the extension function

The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part identifies the extension function as a Java function. It does so by associating the extension function with an in-scope namespace declaration, the URI of which must begin with `java:` (*see below for examples*). The namespace declaration should identify a Java class, for example: `xmlns:myns="java:java.lang.Math"`. However, it could also simply be: `xmlns:myns="java"` (without a colon), with the identification of the Java class being left to the `fname()` part of the extension function.
- The `fname()` part identifies the Java method being called, and supplies the arguments for the method (*see below for examples*). However, if the namespace URI identified by the `prefix:` part does not identify a Java class (*see preceding point*), then the Java class

should be identified in the `fname()` part, before the class and separated from the class by a period (*see the second XSLT example below*).

Note: The class being called must be on the classpath of the machine.

XSLT example

Here are two examples of how a static method can be called. In the first example, the class name (`java.lang.Math`) is included in the namespace URI and, therefore, must not be in the `fname()` part. In the second example, the `prefix:` part supplies the prefix `java:` while the `fname()` part identifies the class as well as the method.

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jmath="java"
              select="jmath:java.lang.Math.cos(3.14)" />
```

The method named in the extension function (`cos()` in the example above) must match the name of a public static method in the named Java class (`java.lang.Math` in the example above).

XQuery example

Here is an XQuery example similar to the XSLT example above:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

User-defined Java classes

If you have created your own Java classes, methods in these classes are called differently according to: (i) whether the classes are accessed via a JAR file or a class file, and (ii) whether these files (JAR or class) are located in the current directory (the same directory as the XSLT or XQuery document) or not. How to locate these files is described in the sections User-Defined Class Files and User-Defined Jar Files. Note that paths to class files not in the current directory and to all JAR files must be specified.

User-Defined Class Files

If access is via a class file, then there are two possibilities:

- The class file is in a package. The XSLT or XQuery file is in the same folder as the Java package. (*See example below.*)

 © 2015 Altova GmbH

- The class file is not packaged. The XSLT or XQuery file is in the same folder as the class file. (*See example below.*)
- The class file is in a package. The XSLT or XQuery file is at some random location. (*See example below.*)
- The class file is not packaged. The XSLT or XQuery file is at some random location. (*See example below.*)

Consider the case where the class file is not packaged and is in the same folder as the XSLT or XQuery document. In this case, since all classes in the folder are found, the file location does not need to be specified. The syntax to identify a class is:

```
java:classname
```

where

`java:` indicates that a user-defined Java function is being called; (Java classes in the current directory will be loaded by default)

`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call.

Class file packaged, XSLT/XQuery file in same folder as Java package

The example below calls the `getVehicleType()`method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is also in the folder `JavaProject`.

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="java:com.altova.extfunc.Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
   <a>
   <xsl:value-of select="car:getVehicleType()"/>
   </a>
</xsl:template>

</xsl:stylesheet>
```

Class file not packaged, XSLT/XQuery file in same folder as class file

The example below calls the `getVehicleType()`method of the `Car` class of the `com.altova.extfunc` package. The `Car` class file is in the following folder location: `JavaProject/com/altova/extfunc`. The XSLT file is also in the folder `JavaProject/com/altova/extfunc`.

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="java:Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
   <a>
   <xsl:value-of select="car:getVehicleType()"/>
   </a>
</xsl:template>

</xsl:stylesheet>
```

Class file packaged, XSLT/XQuery file at any location

The example below calls the `getCarColor()`method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is at any location. In this case, the location of the package must be specified within the URI as a query string. The syntax is:

```
java:classname[?path=uri-of-package]
```

where

`java`: indicates that a user-defined Java function is being called
`uri-of-package` is the URI of the Java package
`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="java:com.altova.extfunc.Car?path=file:///C:/
JavaProject/" >

<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
   <xsl:variable name="myCar" select="car:new('red')" />
   <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
</xsl:template>

</xsl:stylesheet>
```

 © 2015 Altova GmbH

Class file not packaged, XSLT/XQuery file at any location

The example below calls the `getCarColor()`method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is at any location. The location of the class file is specified within the namespace URI as a query string. The syntax is:

```
java:classname[?path=uri-of-classfile]
```

where

`java`: indicates that a user-defined Java function is being called
`uri-of-classfile` is the URI of the folder containing the class file
`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="java:Car?path=file:///C:/JavaProject/com/altova/
extfunc/" >

<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
   <xsl:variable name="myCar" select="car:new('red')" />
   <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
</xsl:template>

</xsl:stylesheet>
```

Note: When a path is supplied via the extension function, the path is added to the ClassLoader.

User-Defined Jar Files

If access is via a JAR file, the URI of the JAR file must be specified using the following syntax:

```
xmlns:classNS="java:classname?path=jar:uri-of-jarfile!/"
```

The method is then called by using the prefix of the namespace URI that identifies the class: `classNS:method()`

In the above:

`java:` indicates that a Java function is being called
`classname` is the name of the user-defined class
`?` is the separator between the classname and the path
`path=jar:` indicates that a path to a JAR file is being given
`uri-of-jarfile` is the URI of the jar file
`!/` is the end delimiter of the path
`classNS:method()` is the call to the method

Alternatively, the classname can be given with the method call. Here are two examples of the syntax:

```
            xmlns:ns1="java:docx.layout.pages?path=jar:file:///c:/projects/
docs/docx.jar!/"
            ns1:main()

            xmlns:ns2="java?path=jar:file:///c:/projects/docs/docx.jar!/"
            ns2:docx.layout.pages.main()
```

Here is a complete XSLT example that uses a JAR file to call a Java extension function:

```
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:fn="http://www.w3.org/2005/xpath-functions"
      xmlns:car="java?path=jar:file:///C:/test/Car1.jar!/" >
<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
      <xsl:variable name="myCar" select="car:Car1.new('red')" />
      <a><xsl:value-of select="car:Car1.getCarColor($myCar)"/></a>
</xsl:template>

<xsl:template match="car"/>

</xsl:stylesheet>
```

Note: When a path is supplied via the extension function, the path is added to the ClassLoader.

Java: Constructors

An extension function can be used to call a Java constructor. All constructors are called with the pseudo-function `new()`.

If the result of a Java constructor call can be implicitly converted to XPath/XQuery datatypes, then the Java extension function will return a sequence that is an XPath/XQuery datatype. If the result of a Java constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped Java object with a type that is the name of the class returning that

 © 2015 Altova GmbH

Java object. For example, if a constructor for the class `java.util.Date` is called (`java.util.Date.new()`), then an object having a type `java.util.Date` is returned. The lexical format of the returned object may not match the lexical format of an XPath datatype and the value would therefore need to be converted to the lexical format of the required XPath datatype and then to the required XPath datatype.

There are two things that can be done with a Java object created by a constructor:

- It can be assigned to a variable:
```
<xsl:variable name="currentdate" select="date:new()"
xmlns:date="java:java.util.Date" />
```
- It can be passed to an extension function (*see Instance Method and Instance Fields*):
```
<xsl:value-of select="date:toString(date:new())"
xmlns:date="java:java.util.Date" />
```

Java: Static Methods and Static Fields

A static method is called directly by its Java name and by supplying the arguments for the method. Static fields (methods that take no arguments), such as the constant-value fields `E` and `PI`, are accessed without specifying any argument.

XSLT examples

Here are some examples of how static methods and fields can be called:

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos( jMath:PI() )" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:E() * jMath:cos(3.14)" />
```

Notice that the extension functions above have the form `prefix:fname()`. The prefix in all three cases is `jMath:`, which is associated with the namespace URI `java:java.lang.Math`. (The namespace URI must begin with `java:`. In the examples above it is extended to contain the class name (`java.lang.Math`).) The `fname()` part of the extension functions must match the name of a public class (e.g. `java.lang.Math`) followed by the name of a public static method with its argument/s (such as `cos(3.14)`) or a public static field (such as `PI()`).

In the examples above, the class name has been included in the namespace URI. If it were not contained in the namespace URI, then it would have to be included in the `fname()` part of the extension function. For example:

```
<xsl:value-of xmlns:java="java:"
              select="java:java.lang.Math.cos(3.14)" />
```

XQuery example

A similar example in XQuery would be:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

Java: Instance Methods and Instance Fields

An instance method has a Java object passed to it as the first argument of the method call. Such a Java object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="1.0" exclude-result-prefixes="date"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="java:java.util.Date"
  xmlns:jlang="java:java.lang">
  <xsl:param name="CurrentDate" select="date:new()"/>
  <xsl:template match="/">
      <enrollment institution-id="Altova School"
                date="{date:toString($CurrentDate)}"
                type="
{jlang:Object.toString(jlang:Object.getClass( date:new() ))}">
      </enrollment>
  </xsl:template>
</xsl:stylesheet>
```

In the example above, the value of the node `enrollment/@type` is created as follows:

1. An object is created with a constructor for the class `java.util.Date` (with the `date:new()` constructor).
2. This Java object is passed as the argument of the `jlang.Object.getClass` method.
3. The object obtained by the `getClass` method is passed as the argument to the `jlang.Object.toString` method.

The result (the value of `@type`) will be a string having the value: `java.util.Date`.

An instance field is theoretically different from an instance method in that it is not a Java object per se that is passed as an argument to the instance field. Instead, a parameter or variable is passed as the argument. However, the parameter/variable may itself contain the value returned by a Java object. For example, the parameter `CurrentDate` takes the value returned by a constructor for the class `java.util.Date`. This value is then passed as an argument to the instance method `date:toString` in order to supply the value of `/enrollment/@date`.

Datatypes: XPath/XQuery to Java

When a Java function is called from within an XPath/XQuery expression, the datatype of the function's arguments is important in determining which of multiple Java classes having the same name is called.

 © 2015 Altova GmbH

In Java, the following rules are followed:

- If there is more than one Java method with the same name, but each has a different number of arguments than the other/s, then the Java method that best matches the number of arguments in the function call is selected.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding Java datatype. If the supplied XPath/XQuery type can be converted to more than one Java type (for example, `xs:integer`), then that Java type is selected which is declared for the selected method. For example, if the Java method being called is `fx(decimal)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to Java's `decimal` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to Java datatypes.

`xs:string`	`java.lang.String`
`xs:boolean`	`boolean` (primitive), `java.lang.Boolean`
`xs:integer`	int, long, short, byte, float, double, and the wrapper classes of these, such as `java.lang.Integer`
`xs:float`	`float` (primitive), `java.lang.Float`, `double` (primitive)
`xs:double`	`double` (primitive), `java.lang.Double`
`xs:decimal`	`float` (primitive), `java.lang.Float`, `double`(primitive), `java.lang.Double`

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the Java type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct Java method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error. However, note that in some cases, it might be possible to create the required Java type by using a Java constructor.

Datatypes: Java to XPath/XQuery

When a Java method returns a value, the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, Java's `java.lang.Boolean` and `boolean` datatypes are converted to `xsd:boolean`.

One-dimensional arrays returned by functions are expanded to a sequence. Multi-dimensional arrays will not be converted, and should therefore be wrapped.

When a wrapped Java object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a Java method (e.g `toString`) to convert the Java object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

.NET Extension Functions

If you are working on the .NET platform on a Windows machine, you can use extension functions written in any of the .NET languages (for example, C#). A .NET extension function can be used within an XPath or XQuery expression to invoke a constructor, property, or method (static or instance) within a .NET class.

A property of a .NET class is called using the syntax `get_PropertyName()`.

This section is organized into the following sub-sections:

- .NET: Constructors
- .NET: Static Methods and Static Fields
- .NET: Instance Methods and Instance Fields
- Datatypes: XPath/XQuery to .NET
- Datatypes: .NET to XPath/XQuery

Form of the extension function

The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part is associated with a URI that identifies the .NET class being addressed.
- The `fname()` part identifies the constructor, property, or method (static or instance) within the .NET class, and supplies any argument/s, if required.
- The URI must begin with `clitype:` (which identifies the function as being a .NET extension function).
- The `prefix:fname()` form of the extension function can be used with system classes and with classes in a loaded assembly. However, if a class needs to be loaded, additional parameters containing the required information will have to be supplied.

 © 2015 Altova GmbH

Parameters

To load an assembly, the following parameters are used:

`asm`	The name of the assembly to be loaded.
`ver`	The version number (maximum of four integers separated by periods).
`sn`	The key token of the assembly's strong name (16 hex digits).
`from`	A URI that gives the location of the assembly (DLL) to be loaded. If the URI is relative, it is relative to the XSLT or XQuery document. If this parameter is present, any other parameter is ignored.
`partialname`	The partial name of the assembly. It is supplied to `Assembly.LoadWith.PartialName()`, which will attempt to load the assembly. If `partialname` is present, any other parameter is ignored.
`loc`	The locale, for example, `en-US`. The default is `neutral`.

If the assembly is to be loaded from a DLL, use the `from` parameter and omit the `sn` parameter. If the assembly is to be loaded from the Global Assembly Cache (GAC), use the `sn` parameter and omit the `from` parameter.

A question mark must be inserted before the first parameter, and parameters must be separated by a semi-colon. The parameter name gives its value with an equals sign (*see example below*).

Examples of namespace declarations

An example of a namespace declaration in XSLT that identifies the system class `System.Environment`:

```
xmlns:myns="clitype:System.Environment"
```

An example of a namespace declaration in XSLT that identifies the class to be loaded as `Trade.Forward.Scrip`:

```
xmlns:myns="clitype:Trade.Forward.Scrip?asm=forward;version=10.6.2.1"
```

An example of a namespace declaration in XQuery that identifies the system class `MyManagedDLL.testClass`:. Two cases are distinguished:

1. When the assembly is loaded from the GAC:

```
declare namespace cs="clitype:MyManagedDLL.testClass?asm=MyManagedDLL;
       ver=1.2.3.4;loc=neutral;sn=b9f091b72dccfba8";
```

2. When the assembly is loaded from the DLL (complete and partial references below):

```
       declare namespace cs="clitype:MyManagedDLL.testClass?from=file:///
C:/Altova
       Projects/extFunctions/MyManagedDLL.dll;

   declare namespace cs="clitype:MyManagedDLL.testClass?
   from=MyManagedDLL.dll;
```

XSLT example

Here is a complete XSLT example that calls functions in system class `System.Math`:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions">
   <xsl:output method="xml" omit-xml-declaration="yes" />
   <xsl:template match="/">
      <math xmlns:math="clitype:System.Math">
         <sqrt><xsl:value-of select="math:Sqrt(9)"/></sqrt>
         <pi><xsl:value-of select="math:PI()"/></pi>
         <e><xsl:value-of select="math:E()"/></e>
         <pow><xsl:value-of select="math:Pow(math:PI(), math:E())"/></pow>
      </math>
   </xsl:template>
</xsl:stylesheet>
```

The namespace declaration on the element `math` associates the prefix `math:` with the URI `clitype:System.Math`. The `clitype:` beginning of the URI indicates that what follows identifies either a system class or a loaded class. The `math:` prefix in the XPath expressions associates the extension functions with the URI (and, by extension, the class) `System.Math`. The extension functions identify methods in the class `System.Math` and supply arguments where required.

XQuery example

Here is an XQuery example fragment similar to the XSLT example above:

```
<math xmlns:math="clitype:System.Math">
   {math:Sqrt(9)}
</math>
```

As with the XSLT example above, the namespace declaration identifies the .NET class, in this case a system class. The XQuery expression identifies the method to be called and supplies the argument.

.NET: Constructors

An extension function can be used to call a .NET constructor. All constructors are called with the pseudo-function `new()`. If there is more than one constructor for a class, then the constructor that most closely matches the number of arguments supplied is selected. If no constructor is deemed to match the supplied argument/s, then a `'No constructor found'` error is returned.

© 2015 Altova GmbH

Constructors that return XPath/XQuery datatypes

If the result of a .NET constructor call can be implicitly converted to XPath/XQuery datatypes, then the .NET extension function will return a sequence that is an XPath/XQuery datatype.

Constructors that return .NET objects

If the result of a .NET constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped .NET object with a type that is the name of the class returning that object. For example, if a constructor for the class `System.DateTime` is called (with `System.DateTime.new()`), then an object having a type `System.DateTime` is returned.

The lexical format of the returned object may not match the lexical format of a required XPath datatype. In such cases, the returned value would need to be: (i) converted to the lexical format of the required XPath datatype; and (ii) cast to the required XPath datatype.

There are three things that can be done with a .NET object created by a constructor:

- It can be used within a variable:
  ```
  <xsl:variable name="currentdate" select="date:new(2008, 4, 29)"
  xmlns:date="clitype:System.DateTime" />
  ```
- It can be passed to an extension function (*see Instance Method and Instance Fields*):
  ```
  <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
  xmlns:date="clitype:System.DateTime" />
  ```
- It can be converted to a string, number, or boolean:
- ```
  <xsl:value-of select="xs:integer(data:get_Month(date:new(2008, 4, 29)))"
  xmlns:date="clitype:System.DateTime" />
  ```

.NET: Static Methods and Static Fields

A static method is called directly by its name and by supplying the arguments for the method. The name used in the call must exactly match a public static method in the class specified. If the method name and the number of arguments that were given in the function call matches more than one method in a class, then the types of the supplied arguments are evaluated for the best match. If a match cannot be found unambiguously, an error is reported.

Note: A field in a .NET class is considered to be a method without any argument. A property is called using the syntax `get_PropertyName()`.

Examples

An XSLT example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<xsl:value-of select="math:Sin(30)" xmlns:math="clitype:System.Math"/>
```
```

An XSLT example showing a call to a field (considered a method with no argument) (`System.Double.MaxValue()`):

```
<xsl:value-of select="double:MaxValue()" xmlns:double="clitype:System.Double"/>
```

An XSLT example showing a call to a property (syntax is `get_PropertyName()`) (`System.String()`):

```
<xsl:value-of select="string:get_Length('my string')"
xmlns:string="clitype:System.String"/>
```

An XQuery example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<sin xmlns:math="clitype:System.Math">
   { math:Sin(30) }
</sin>
```

.NET: Instance Methods and Instance Fields

An instance method has a .NET object passed to it as the first argument of the method call. This .NET object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions">
   <xsl:output method="xml" omit-xml-declaration="yes"/>
   <xsl:template match="/">
      <xsl:variable name="releasedate"
         select="date:new(2008, 4, 29)"
         xmlns:date="clitype:System.DateTime"/>
      <doc>
         <date>
            <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
               xmlns:date="clitype:System.DateTime"/>
         </date>
         <date>
            <xsl:value-of select="date:ToString($releasedate)"
               xmlns:date="clitype:System.DateTime"/>
         </date>
      </doc>
   </xsl:template>
</xsl:stylesheet>
```

In the example above, a `System.DateTime` constructor (`new(2008, 4, 29)`) is used to create a .NET object of type `System.DateTime`. This object is created twice, once as the value of the variable `releasedate`, a second time as the first and only argument of the `System.DateTime.ToString()` method. The instance method `System.DateTime.ToString()` is called twice, both times with the `System.DateTime` constructor (`new(2008, 4, 29)`) as its first

 © 2015 Altova GmbH

and only argument. In one of these instances, the variable `releasedate` is used to get the .NET object.

Instance methods and instance fields

The difference between an instance method and an instance field is theoretical. In an instance method, a .NET object is directly passed as an argument; in an instance field, a parameter or variable is passed instead—though the parameter or variable may itself contain a .NET object. For example, in the example above, the variable `releasedate` contains a .NET object, and it is this variable that is passed as the argument of `ToString()` in the second `date` element constructor. Therefore, the `ToString()` instance in the first `date` element is an instance method while the second is considered to be an instance field. The result produced in both instances, however, is the same.

Datatypes: XPath/XQuery to .NET

When a .NET extension function is used within an XPath/XQuery expression, the datatypes of the function's arguments are important for determining which one of multiple .NET methods having the same name is called.

In .NET, the following rules are followed:

- If there is more than one method with the same name in a class, then the methods available for selection are reduced to those that have the same number of arguments as the function call.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding .NET datatype. If the supplied XPath/XQuery type can be converted to more than one .NET type (for example, `xs:integer`), then that .NET type is selected which is declared for the selected method. For example, if the .NET method being called is `fx(double)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to .NET's `double` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to .NET datatypes.

`xs:string`	`StringValue, string`
`xs:boolean`	`BooleanValue, bool`
`xs:integer`	`IntegerValue, decimal, long, integer, short, byte, double, float`
`xs:float`	`FloatValue, float, double`
`xs:double`	`DoubleValue, double`
`xs:decimal`	`DecimalValue, decimal, double, float`

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the .NET type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct .NET method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error.

Datatypes: .NET to XPath/XQuery

When a .NET method returns a value and the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, .NET's `decimal` datatype is converted to `xsd:decimal`.

When a .NET object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a .NET method (for example `System.DateTime.ToString()`) to convert the .NET object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

MSXSL Scripts for XSLT

The `<msxsl:script>` element contains user-defined functions and variables that can be called from within XPath expressions in the XSLT stylesheet. The `<msxsl:script>` is a top-level element, that is, it must be a child element of `<xsl:stylesheet>` or `<xsl:transform>`.

The `<msxsl:script>` element must be in the namespace `urn:schemas-microsoft-com:xslt` (*see example below*).

Scripting language and namespace

The scripting language used within the block is specified in the `<msxsl:script>` element's `language` attribute and the namespace to be used for function calls from XPath expressions is identified with the `implements-prefix` attribute (*see below*).

 © 2015 Altova GmbH

```
<msxsl:script language="scripting-language" implements-prefix="user-namespace-
prefix">

   function-1 or variable-1
   ...
   function-n or variable-n

</msxsl:script>
```

The <msxsl:script> element interacts with the Windows Scripting Runtime, so only languages that are installed on your machine may be used within the <msxsl:script> element. **The .NET Framework 2.0 platform or higher must be installed for MSXSL scripts to be used.** Consequently, the .NET scripting languages can be used within the <msxsl:script> element.

The language attribute accepts the same values as the language attribute on the HTML <script> element. If the language attribute is not specified, then Microsoft JScript is assumed as the default.

The implements-prefix attribute takes a value that is a prefix of a declared in-scope namespace. This namespace typically will be a user namespace that has been reserved for a function library. All functions and variables defined within the <msxsl:script> element will be in the namespace identified by the prefix specified in the implements-prefix attribute. When a function is called from within an XPath expression, the fully qualified function name must be in the same namespace as the function definition.

Example

Here is an example of a complete XSLT stylesheet that uses a function defined within a <msxsl:script> element.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:msxsl="urn:schemas-microsoft-com:xslt"
   xmlns:user="http://mycompany.com/mynamespace">

  <msxsl:script language="VBScript" implements-prefix="user">
    <![CDATA[
    ' Input: A currency value: the wholesale price
    ' Returns: The retail price: the input value plus 20% margin,
    ' rounded to the nearest cent
    dim a as integer  = 13
    Function AddMargin(WholesalePrice) as integer
      AddMargin = WholesalePrice * 1.2 + a
    End Function
  ]]>
  </msxsl:script>

  <xsl:template match="/">
    <html>
```

```
      <body>
        <p>
          <b>Total Retail Price =
            $<xsl:value-of select="user:AddMargin(50)"/>
          </b>
          <br/>
          <b>Total Wholesale Price =
            $<xsl:value-of select="50"/>
          </b>
        </p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Datatypes

The values of parameters passed into and out of the script block are limited to XPath datatypes. This restriction does not apply to data passed among functions and variables within the script block.

Assemblies

An assembly can be imported into the script by using the **msxsl:assembly** element. The assembly is identified via a name or a URI. The assembly is imported when the stylesheet is compiled. Here is a simple representation of how the `msxsl:assembly` element is to be used.

```
<msxsl:script>
       <msxsl:assembly name="myAssembly.assemblyName" />
       <msxsl:assembly href="pathToAssembly" />

       ...

</msxsl:script>
```

The assembly name can be a full name, such as:

```
"system.Math, Version=3.1.4500.1 Culture=neutral
PublicKeyToken=a46b3f648229c514"
```

or a short name, such as `"myAssembly.Draw"`.

Namespaces

Namespaces can be declared with the **msxsl:using** element. This enables assembly classes to be written in the script without their namespaces, thus saving you some tedious typing. Here is how the `msxsl:using` element is used so as to declare namespaces.

© 2015 Altova GmbH

```
<msxsl:script>
       <msxsl:using namespace="myAssemblyNS.NamespaceName" />

       ...

</msxsl:script>
```

The value of the `namespace` attribute is the name of the namespace.

11.2 Technical Data

This section contains useful background information on the technical aspects of your software. It is organized into the following sections:

- OS and Memory Requirements
- Altova XML Validator
- Altova XSLT and XQuery Engines
- Unicode Support
- Internet Usage

11.2.1 OS and Memory Requirements

Operating System
Altova software applications are available for the following platforms:

- 32-bit Windows applications for Windows XP, Windows Vista, Windows 7, Windows 8, Windows Server 2003 and 2008
- 64-bit Windows applications for Windows Vista, Windows 7, Windows 8, Windows Server 2012

Memory
Since the software is written in C++ it does not require the overhead of a Java Runtime Environment and typically requires less memory than comparable Java-based applications. However, each document is loaded fully into memory so as to parse it completely and to improve viewing and editing speed. The memory requirement increases with the size of the document.

Memory requirements are also influenced by the unlimited Undo history. When repeatedly cutting and pasting large selections in large documents, available memory can rapidly be depleted.

11.2.2 Altova XML Validator

When opening any XML document, the application uses its built-in XML validator to check for well-formedness, validate the document against a schema (if specified), and build trees and infosets. The XML validator is also used to provide intelligent editing help while you edit documents and to dynamically display any validation error that may occur.

The built-in XML validator implements the Final Recommendation of the W3C's XML Schema 1.0 and 1.1 specification. New developments recommended by the W3C's XML Schema Working Group are continuously being incorporated in the XML validator, so that Altova products give you a state-of-the-art development environment.

11.2.3 Altova XSLT and XQuery Engines

Altova products use the Altova XSLT 1.0, 2.0, and 3.0 Engines and the Altova XQuery 1.0 and 3.0 Engines. Documentation about implementation-specific behavior for each engine is in the

 © 2015 Altova GmbH

appendices of the documentation (Engine Information), should that engine be used in the product.

Note: Altova MapForce generates code using the XSLT 1.0, 2.0 and XQuery 1.0 engines.

11.2.4 Unicode Support

Altova's XML products provide full Unicode support. To edit an XML document, you will also need a font that supports the Unicode characters being used by that document.

Please note that most fonts only contain a very specific subset of the entire Unicode range and are therefore typically targeted at the corresponding writing system. If some text appears garbled, the reason could be that the font you have selected does not contain the required glyphs. So it is useful to have a font that covers the entire Unicode range, especially when editing XML documents in different languages or writing systems. A typical Unicode font found on Windows PCs is Arial Unicode MS.

In the `/Examples` folder of your application folder you will find an XHTML file called `UnicodeUTF-8.html` that contains the following sentence in a number of different languages and writing systems:

- *When the world wants to talk, it speaks Unicode*
- *Wenn die Welt miteinander spricht, spricht sie Unicode*
- 世界的に話すなら、Unicode です。

Opening this XHTML file will give you a quick impression of Unicode's possibilities and also indicate what writing systems are supported by the fonts available on your PC.

11.2.5 Internet Usage

Altova applications will initiate Internet connections on your behalf in the following situations:

- If you click the "Request evaluation key-code" in the Registration dialog (**Help | Software Activation**), the three fields in the registration dialog box are transferred to our web server by means of a regular http (port 80) connection and the free evaluation key-code is sent back to the customer via regular SMTP e-mail.
- In some Altova products, you can open a file over the Internet (**File | Open | Switch to URL**). In this case, the document is retrieved using one of the following protocol methods and connections: HTTP (normally port 80), FTP (normally port 20/21), HTTPS (normally port 443). You could also run an HTTP server on port 8080. (In the URL dialog, specify the port after the server name and a colon.)
- If you open an XML document that refers to an XML Schema or DTD and the document is specified through a URL, the referenced schema document is also retrieved through a HTTP connection (port 80) or another protocol specified in the URL (see Point 2 above). A schema document will also be retrieved when an XML file is validated. Note that validation might happen automatically upon opening a document if you have instructed the application to do this (in the File tab of the Options dialog (**Tools | Options**)).
- In Altova applications using WSDL and SOAP, web service connections are defined by the WSDL documents.
- If you are using the **Send by Mail** command (**File | Send by Mail**) in XMLSpy, the

current selection or file is sent by means of any MAPI-compliant mail program installed on the user's PC.

- As part of Software Activation and LiveUpdate as further described in the Altova Software License Agreement.

© 2015 Altova GmbH

11.3 License Information

This section contains:

- Information about the distribution of this software product
- Information about software activation and license metering
- Information about the intellectual property rights related to this software product
- The End-User License Agreement governing the use of this software product

Please read this information carefully. It is binding upon you since you agreed to these terms when you installed this software product.

11.3.1 Electronic Software Distribution

This product is available through electronic software distribution, a distribution method that provides the following unique benefits:

- You can evaluate the software free-of-charge before making a purchasing decision.
- Once you decide to buy the software, you can place your order online at the Altova website and immediately get a fully licensed product within minutes.
- When you place an online order, you always get the latest version of our software.
- The product package includes a comprehensive integrated onscreen help system. The latest version of the user manual is available at www.altova.com (i) in HTML format for online browsing, and (ii) in PDF format for download (and to print if you prefer to have the documentation on paper).

30-day evaluation period
After downloading this product, you can evaluate it for a period of up to 30 days free of charge. About 20 days into this evaluation period, the software will start to remind you that it has not yet been licensed. The reminder message will be displayed once each time you start the application. If you would like to continue using the program after the 30-day evaluation period, you have to purchase an Altova Software License Agreement, which is delivered in the form of a key-code that you enter into the Software Activation dialog to unlock the product. You can purchase your license at the online shop at the Altova website.

Helping Others within Your Organization to Evaluate the Software
If you wish to distribute the evaluation version within your company network, or if you plan to use it on a PC that is not connected to the Internet, you may only distribute the Setup programs, provided that they are not modified in any way. Any person that accesses the software installer that you have provided, must request their own 30-day evaluation license key code and after expiration of their evaluation period, must also purchase a license in order to be able to continue using the product.

For further details, please refer to the Altova Software License Agreement at the end of this section.

11.3.2 Software Activation and License Metering

As part of Altova's Software Activation, the software may use your internal network and Internet connection for the purpose of transmitting license-related data at the time of installation, registration, use, or update to an Altova-operated license server and validating the authenticity of the license-related data in order to protect Altova against unlicensed or illegal use of the software and to improve customer service. Activation is based on the exchange of license related data such as operating system, IP address, date/time, software version, and computer name, along with other information between your computer and an Altova license server.

Your Altova product has a built-in license metering module that further helps you avoid any unintentional violation of the End User License Agreement. Your product is licensed either as a single-user or multi-user installation, and the license-metering module makes sure that no more than the licensed number of users use the application concurrently.

This license-metering technology uses your local area network (LAN) to communicate between instances of the application running on different computers.

Single license
When the application starts up, as part of the license metering process, the software sends a short broadcast datagram to find any other instance of the product running on another computer in the same network segment. If it doesn't get any response, it will open a port for listening to other instances of the application.

Multi license
If more than one instance of the application is used within the same LAN, these instances will briefly communicate with each other on startup. These instances exchange key-codes in order to help you to better determine that the number of concurrent licenses purchased is not accidentally violated. This is the same kind of license metering technology that is common in the Unix world and with a number of database development tools. It allows Altova customers to purchase reasonably-priced concurrent-use multi-user licenses.

We have also designed the applications so that they send few and small network packets so as to not put a burden on your network. The TCP/IP ports (2799) used by your Altova product are officially registered with the IANA (*see* the IANA website (http://www.iana.org/) *for details*) and our license-metering module is tested and proven technology.

If you are using a firewall, you may notice communications on port 2799 between the computers that are running Altova products. You are, of course, free to block such traffic between different groups in your organization, as long as you can ensure by other means, that your license agreement is not violated.

 © 2015 Altova GmbH

You will also notice that, if you are online, your Altova product contains many useful functions; these are unrelated to the license-metering technology.

11.3.3 Intellectual Property Rights

The Altova Software and any copies that you are authorized by Altova to make are the intellectual property of and are owned by Altova and its suppliers. The structure, organization and code of the Software are the valuable trade secrets and confidential information of Altova and its suppliers. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. Altova retains the ownership of all patents, copyrights, trade secrets, trademarks and other intellectual property rights pertaining to the Software, and that Altova's ownership rights extend to any images, photographs, animations, videos, audio, music, text and "applets" incorporated into the Software and all accompanying printed materials. Notifications of claimed copyright infringement should be sent to Altova's copyright agent as further provided on the Altova Web Site.

Altova software contains certain Third Party Software that is also protected by intellectual property laws, including without limitation applicable copyright laws as described in detail at http://www.altova.com/legal_3rdparty.html.

All other names or trademarks are the property of their respective owners.

11.3.4 Altova End User License Agreement

THIS IS A LEGAL DOCUMENT -- RETAIN FOR YOUR RECORDS

ALTOVA® END USER LICENSE AGREEMENT

Licensor:
Altova GmbH
Rudolfsplatz 13a/9
A-1010 Wien
Austria

Important - Read Carefully. Notice to User:

This End User License Agreement ("Agreement") is a legal document between you and Altova GmbH ("Altova"). It is important that you read this document before using the Altova-provided software ("Software") and any accompanying documentation, including, without limitation printed materials, 'online' files, or electronic documentation ("Documentation"). By clicking the "I accept" and "Next" buttons below, or by installing, or otherwise using the Software, you agree to be bound by the terms of this Agreement as well as the Altova Privacy Policy ("Privacy Policy") including, without limitation, the warranty disclaimers, limitation of liability, data use and termination provisions below, whether or not you decide to purchase the Software. You agree that this agreement is enforceable like any written agreement negotiated and signed by you. If you do not agree, you are not licensed to use the Software, and you must destroy any downloaded copies of the

Software in your possession or control. You may print a copy of this Agreement as part of the installation process at the time of acceptance. Alternatively, a copy of this Agreement may be found at http://www.altova.com/eula and a copy of the Privacy Policy may be found at http://www.altova.com/privacy.

1. SOFTWARE LICENSE

(a) License Grant.

(i) Upon your acceptance of this Agreement Altova grants you a non-exclusive, non-transferable (except as provided below), limited license, without the right to grant sublicenses, to install and use a copy of the Software on one compatible personal computer or workstation in the same local area network (LAN) up to the Permitted Number of computers. Subject to the limitations set forth in Section 1(c), you may install and use a copy of the Software on more than one of your compatible personal computers or workstations if you have purchased a Named-User license. Subject to the limitations set forth in Sections 1(d) and 1(e), users may use the software concurrently on a network. The Permitted Number of computers and/or users and the type of license, e.g. Installed, Named-Users, and Concurrent-User, shall be determined and specified at such time as you elect to purchase the Software. Installed user licenses are intended to be fixed and not concurrent. In other words, you cannot uninstall the Software on one machine in order to reinstall that license to a different machine and then uninstall and reinstall back to the original machine. Installations should be static. Notwithstanding the foregoing, permanent uninstallations and redeployments are acceptable in limited circumstances such as if an employee leaves the company or the machine is permanently decommissioned. During the evaluation period, hereinafter defined, only a single user may install and use the software on one (1) personal computer or workstation. If you have licensed the Software as part of a suite of Altova software products (collectively, the "Suite") and have not installed each product individually, then the Agreement governs your use of all of the software included in the Suite.

(ii) If you have licensed SchemaAgent, then the terms and conditions of this Agreement apply to your use of the SchemaAgent server software ("SchemaAgent Server") included therein, as applicable, and you are licensed to use SchemaAgent Server solely in connection with your use of Altova Software and solely for the purposes described in the accompanying documentation.

(iii) If you have licensed Software that enables users to generate source code, your license to install and use a copy of the Software as provided herein permits you to generate source code based on (i) Altova Library modules that are included in the Software (such generated code hereinafter referred to as the "Restricted Source Code") and (ii) schemas or mappings that you create or provide (such code as may be generated from your schema or mapping source materials hereinafter referred to as the "Unrestricted Source Code"). In addition to the rights granted herein, Altova grants you a non-exclusive, non-transferable, limited license to compile the complete generated code (comprised of the combination of the Restricted Source Code and the Unrestricted Source Code) into executable object code form, and to use, copy, distribute or license that executable. You may not distribute or redistribute, sublicense, sell, or transfer the Restricted Source Code to a third-party in the un-compiled form unless said third-party already has a license to the Restricted Source Code through their separate agreement with Altova. Notwithstanding anything to the contrary herein, you may not distribute, incorporate or combine with other software, or otherwise use the Altova Library modules or Restricted Source Code, or any Altova intellectual property embodied in or associated with the Altova Library modules or Restricted Source Code, in any manner that would subject the Restricted Source Code to the terms of a copyleft, free software or open source license that would require the Restricted Source Code or Altova Library modules source code to be disclosed in source code form. Notwithstanding anything to the contrary herein, you may not use the Software to develop and distribute other

software programs that directly compete with any Altova software or service without prior written permission. Altova reserves all other rights in and to the Software. With respect to the feature(s) of UModel that permit reverse-engineering of your own source code or other source code that you have lawfully obtained, such use by you does not constitute a violation of this Agreement. Except as otherwise expressly permitted in Section 1(j) reverse engineering of the Software is strictly prohibited as further detailed therein.

(iv) In the event Restricted Source Code is incorporated into executable object code form, you will include the following statement in (1) introductory splash screens, or if none, within one or more screens readily accessible by the end-user, and (2) in the electronic and/or hard copy documentation: "Portions of this program were developed using Altova® [name of Altova Software, e.g. MapForce® 2016] and includes libraries owned by Altova GmbH, Copyright © 2007-2016 Altova GmbH (www.altova.com)."

(b) Server Use for Installation and Use of SchemaAgent. You may install one (1) copy of the Software on a computer file server within your internal network solely for the purpose of downloading and installing the Software onto other computers within your internal network up to the Permitted Number of computers in a commercial environment only. If you have licensed SchemaAgent, then you may install SchemaAgent Server on any server computer or workstation and use it in connection with your Software. No other network use is permitted, including without limitation using the Software either directly or through commands, data or instructions from or to a computer not part of your internal network, for Internet or Web-hosting services or by any user not licensed to use this copy of the Software through a valid license from Altova.

(c) Named-Use. If you have licensed the "Named-User" version of the software, you may install the Software on up to five (5) compatible personal computers or workstations of which you are the primary user thereby allowing you to switch from one computer to the other as necessary provided that only one (1) instance of the Software will be used by you as the Named-User at any given time. If you have purchased multiple Named-User licenses, each individual Named-User will receive a separate license key code.

(d) Concurrent Use in Same Local Area Network (LAN). If you have licensed a "Concurrent-User" version of the Software, you may install the Software on any compatible computers in a commercial environment only, up to ten (10) times the Permitted Number of users, provided that only the Permitted Number of users actually use the Software at the same time and further provided that the computers on which the Software is installed are on the same local area network (LAN). The Permitted Number of concurrent users shall be delineated at such time as you elect to purchase the Software licenses. Each separate local area network (LAN) requires its own set of separate Concurrent User Licenses for those wishing to use the Concurrent User versions of the Software in more than one location or on more than one network, all subject to the above Permitted Number limitations and based on the number of users using the Software. If a computer is not on the same local area network (LAN), then a locally installed user license or a license dedicated to concurrent use in a virtual environment is required.

(e) Concurrent Use in Virtual Environment. If you have purchased Concurrent-User Licenses, you may install a copy of the Software on a single host terminal server (Microsoft Terminal Server or Citrix Metaframe), application virtualization server (Microsoft App-V, Citrix XenApp, or VMWare ThinApp) or virtual machine environment within your internal network for the sole and exclusive purpose of permitting individual users within your organization to access and use the Software through a terminal server, application virtualization session, or virtual machine environment from another computer provided that the total number of users that access or use the Software concurrently at any given point in time on such network, virtual machine or terminal server does not exceed the Permitted Number; and provided that the total number of users

authorized to use the Software through the terminal server, application virtualization session, or virtual machine environment does not exceed ten (10) times the Permitted Number of users. Key codes for concurrent users cannot be deployed to more than one host terminal server, application virtualization server or virtual machine environment. You must deploy a reliable and accurate means of preventing users from exceeding the Permitted Number of concurrent users. Altova makes no warranties or representations about the performance of Altova software in a terminal server, application virtualization session, or virtual machine environment and the foregoing are expressly excluded from the limited warranty in Section 5 hereof. Technical support is not available with respect to issues arising from use in such environments.

(f) Backup and Archival Copies. You may make one (1) backup and one (1) archival copy of the Software, provided your backup and archival copies are not installed or used on any computer and further provided that all such copies shall bear the original and unmodified copyright, patent and other intellectual property markings that appear on or in the Software. You may not transfer the rights to a backup or archival copy unless you transfer all rights in the Software as provided under Section 3.

(g) Key Codes, Upgrades and Updates. Prior to your purchase and as part of the registration for the thirty (30) day evaluation period, as applicable, you will receive an evaluation key code. You will receive a purchase key code when you elect to purchase the Software from either Altova GmbH or an authorized reseller. The purchase key code will enable you to activate the Software beyond the initial evaluation period. You may not re-license, reproduce or distribute any key code except with the express written permission of Altova. If the Software that you have licensed is an upgrade or an update, then the latest update or upgrade that you download and install replaces all or part of the Software previously licensed. The update or upgrade and the associated license keys does not constitute the granting of a second license to the Software in that you may not use the upgrade or updated copy in addition to the copy of the Software that it is replacing and whose license has terminated.

(h) Title. Title to the Software is not transferred to you. Ownership of all copies of the Software and of copies made by you is vested in Altova, subject to the rights of use granted to you in this Agreement. As between you and Altova, documents, files, stylesheets, generated program code (including the Unrestricted Source Code) and schemas that are authored or created by you via your utilization of the Software, in accordance with its Documentation and the terms of this Agreement, are your property unless they are created using Evaluation Software, as defined in Section 4 of this Agreement, in which case you have only a limited license to use any output that contains generated program code (including Unrestricted Source Code) such as Java, C++, C#, VB.NET or XSLT and associated project files and build scripts, as well as generated XML, XML Schemas, documentation, UML diagrams, and database structures only for the thirty (30) day evaluation period.

(i) Reverse Engineering. Except and to the limited extent as may be otherwise specifically provided by applicable law in the European Union, you may not reverse engineer, decompile, disassemble or otherwise attempt to discover the source code, underlying ideas, underlying user interface techniques or algorithms of the Software by any means whatsoever, directly or indirectly, or disclose any of the foregoing, except to the extent you may be expressly permitted to decompile under applicable law in the European Union, if it is essential to do so in order to achieve operability of the Software with another software program, and you have first requested Altova to provide the information necessary to achieve such operability and Altova has not made such information available. Altova has the right to impose reasonable conditions and to request a reasonable fee before providing such information. Any information supplied by Altova or obtained by you, as permitted hereunder, may only be used by you for the purpose described herein and may not be disclosed to any third party or used to create any software which is

substantially similar to the expression of the Software. Requests for information from users in the European Union with respect to the above should be directed to the Altova Customer Support Department.

(j) Other Restrictions. You may not loan, rent, lease, sublicense, distribute or otherwise transfer all or any portion of the Software to third parties except to the limited extent set forth in Section 3 or as otherwise expressly provided. You may not copy the Software except as expressly set forth above, and any copies that you are permitted to make pursuant to this Agreement must contain the same copyright, patent and other intellectual property markings that appear on or in the Software. You may not modify, adapt or translate the Software. You may not, directly or indirectly, encumber or suffer to exist any lien or security interest on the Software; knowingly take any action that would cause the Software to be placed in the public domain; or use the Software in any computer environment not specified in this Agreement. You may not permit any use of or access to the Software by any third party in connection with a commercial service offering, such as for a cloud-based or web-based SaaS offering.

You will comply with applicable law and Altova's instructions regarding the use of the Software. You agree to notify your employees and agents who may have access to the Software of the restrictions contained in this Agreement and to ensure their compliance with these restrictions.

(k) NO GUARANTEE. THE SOFTWARE IS NEITHER GUARANTEED NOR WARRANTED TO BE ERROR-FREE NOR SHALL ANY LIABILITY BE ASSUMED BY ALTOVA IN THIS RESPECT. NOTWITHSTANDING ANY SUPPORT FOR ANY TECHNICAL STANDARD, THE SOFTWARE IS NOT INTENDED FOR USE IN OR IN CONNECTION WITH, WITHOUT LIMITATION, THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT NAVIGATION, COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL EQUIPMENT, MEDICAL DEVICES OR LIFE SUPPORT SYSTEMS, MEDICAL OR HEALTH CARE APPLICATIONS, OR OTHER APPLICATIONS WHERE THE FAILURE OF THE SOFTWARE OR ERRORS IN DATA PROCESSING COULD LEAD TO DEATH, PERSONAL INJURY OR SEVERE PHYSICAL OR ENVIRONMENTAL DAMAGE. YOU AGREE THAT YOU ARE SOLELY RESPONSIBLE FOR THE ACCURACY AND ADEQUACY OF THE SOFTWARE AND ANY DATA GENERATED OR PROCESSED BY THE SOFTWARE FOR YOUR INTENDED USE AND YOU WILL DEFEND, INDEMNIFY AND HOLD ALTOVA, ITS OFFICERS AND EMPLOYEES HARMLESS FROM ANY THIRD PARTY CLAIMS, DEMANDS, OR SUITS THAT ARE BASED UPON THE ACCURACY AND ADEQUACY OF THE SOFTWARE IN YOUR USE OR ANY DATA GENERATED BY THE SOFTWARE IN YOUR USE.

2. INTELLECTUAL PROPERTY RIGHTS

You acknowledge that the Software and any copies that you are authorized by Altova to make are the intellectual property of and are owned by Altova and its suppliers. The structure, organization and code of the Software are the valuable trade secrets and confidential information of Altova and its suppliers. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. You acknowledge that Altova retains the ownership of all patents, copyrights, trade secrets, trademarks and other intellectual property rights pertaining to the Software, and that Altova's ownership rights extend to any images, photographs, animations, videos, audio, music, text and "applets" incorporated into the Software and all accompanying printed materials. You will take no actions which adversely affect Altova's intellectual property rights in the Software. Trademarks shall be used in accordance with accepted trademark practice, including identification of trademark owners' names. Trademarks may only be used to identify printed output produced by the Software, and such use of any trademark does not give you any right of ownership in that

trademark. Altova®, XMLSpy®, Authentic®, StyleVision®, MapForce®, UModel®, DatabaseSpy®, DiffDog®, SchemaAgent®, SemanticWorks®, MissionKit®, Markup Your Mind®, Nanonull™, RaptorXML™, RaptorXML Server™, RaptorXML +XBRL Server™, Powered By RaptorXML™, FlowForce Server™, StyleVision Server™, and MapForce Server™ are trademarks of Altova GmbH. (pending or registered in numerous countries). Unicode and the Unicode Logo are trademarks of Unicode, Inc. Windows, Windows XP, Windows Vista, Windows 7, and Windows 8 are trademarks of Microsoft. W3C, CSS, DOM, MathML, RDF, XHTML, XML and XSL are trademarks (registered in numerous countries) of the World Wide Web Consortium (W3C); marks of the W3C are registered and held by its host institutions, MIT, INRIA and Keio. Except as expressly stated above, this Agreement does not grant you any intellectual property rights in the Software. Notifications of claimed copyright infringement should be sent to Altova's copyright agent as further provided on the Altova Web Site.

3. LIMITED TRANSFER RIGHTS

Notwithstanding the foregoing, you may transfer all your rights to use the Software to another person or legal entity provided that: (a) you also transfer this Agreement, the Software and all other software or hardware bundled or pre-installed with the Software, including all copies, updates and prior versions, and all copies of font software converted into other formats, to such person or entity; (b) you retain no copies, including backups and copies stored on a computer; (c) the receiving party secures a personalized key code from Altova; and (d) the receiving party accepts the terms and conditions of this Agreement and any other terms and conditions upon which you legally purchased a license to the Software. Notwithstanding the foregoing, you may not transfer education, pre-release, or not-for-resale copies of the Software.

4. PRE-RELEASE AND EVALUATION PRODUCT ADDITIONAL TERMS

If the product you have received with this license is pre-commercial release or beta Software ("Pre-release Software"), then this Section applies. In addition, this section applies to all evaluation and/ or demonstration copies of Altova software ("Evaluation Software") and continues in effect until you purchase a license. To the extent that any provision in this section is in conflict with any other term or condition in this Agreement, this section shall supersede such other term(s) and condition(s) with respect to the Pre-release and/or Evaluation Software, but only to the extent necessary to resolve the conflict. You acknowledge that the Pre-release Software is a pre-release version, does not represent final product from Altova, and may contain bugs, errors and other problems that could cause system or other failures and data loss. CONSEQUENTLY, THE PRE-RELEASE AND/OR EVALUATION SOFTWARE IS PROVIDED TO YOU **"AS-IS" WITH NO WARRANTIES FOR USE OR PERFORMANCE,** AND ALTOVA DISCLAIMS ANY WARRANTY OR LIABILITY OBLIGATIONS TO YOU OF ANY KIND, WHETHER EXPRESS OR IMPLIED. WHERE LEGALLY LIABILITY CANNOT BE EXCLUDED FOR PRE-RELEASE AND/OR EVALUATION SOFTWARE, BUT IT MAY BE LIMITED, ALTOVA'S LIABILITY AND THAT OF ITS SUPPLIERS SHALL BE LIMITED TO THE SUM OF FIFTY DOLLARS (USD $50) IN TOTAL. If the Evaluation Software has a time-out feature, then the software will cease operation after the conclusion of the designated evaluation period. Upon such expiration date, your license will expire unless otherwise extended. Your license to use any output created with the Evaluation Software that contains generated program code (including Unrestricted Source Code) such as Java, C++, C, VB.NET or XSLT and associated project files and build scripts as well as generated XML, XML Schemas, documentation, UML diagrams, and database structures terminates automatically upon the expiration of the designated evaluation period but the license to use such output is revived upon your purchase of a license for the Software that you evaluated and used to create such output. Access to any files created with the Evaluation Software is entirely at your risk. You acknowledge that Altova has not promised or guaranteed to you that Pre-release Software will be announced or made available to anyone in the future, that Altova has no express or implied

obligation to you to announce or introduce the Pre-release Software, and that Altova may not introduce a product similar to or compatible with the Pre-release Software. Accordingly, you acknowledge that any research or development that you perform regarding the Pre-release Software or any product associated with the Pre-release Software is done entirely at your own risk. During the term of this Agreement, if requested by Altova, you will provide feedback to Altova regarding testing and use of the Pre-release Software, including error or bug reports. If you have been provided the Pre-release Software pursuant to a separate written agreement, your use of the Software is governed by such agreement. You may not sublicense, lease, loan, rent, distribute or otherwise transfer the Pre-release Software. Upon receipt of a later unreleased version of the Pre-release Software or release by Altova of a publicly released commercial version of the Software, whether as a stand-alone product or as part of a larger product, you agree to return or destroy all earlier Pre-release Software received from Altova and to abide by the terms of the license agreement for any such later versions of the Pre-release Software.

5. LIMITED WARRANTY AND LIMITATION OF LIABILITY

(a) Limited Warranty and Customer Remedies. Altova warrants to the person or entity that first purchases a license for use of the Software pursuant to the terms of this Agreement that (i) the Software will perform substantially in accordance with any accompanying Documentation for a period of ninety (90) days from the date of receipt, and (ii) any support services provided by Altova shall be substantially as described in Section 6 of this agreement. Some states and jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you. To the extent allowed by applicable law, implied warranties on the Software, if any, are limited to ninety (90) days. Altova's and its suppliers' entire liability and your exclusive remedy shall be, at Altova's option, either (i) return of the price paid, if any, or (ii) repair or replacement of the Software that does not meet Altova's Limited Warranty and which is returned to Altova with a copy of your receipt. This Limited Warranty is void if failure of the Software has resulted from accident, abuse, misapplication, abnormal use, Trojan horse, virus, or any other malicious external code. Any replacement Software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer. This limited warranty does not apply to Evaluation and/or Pre-release Software.

(b) No Other Warranties and Disclaimer. THE FOREGOING LIMITED WARRANTY AND REMEDIES STATE THE SOLE AND EXCLUSIVE REMEDIES FOR ALTOVA OR ITS SUPPLIER'S BREACH OF WARRANTY. ALTOVA AND ITS SUPPLIERS DO NOT AND CANNOT WARRANT THE PERFORMANCE OR RESULTS YOU MAY OBTAIN BY USING THE SOFTWARE. EXCEPT FOR THE FOREGOING LIMITED WARRANTY, AND FOR ANY WARRANTY, CONDITION, REPRESENTATION OR TERM TO THE EXTENT WHICH THE SAME CANNOT OR MAY NOT BE EXCLUDED OR LIMITED BY LAW APPLICABLE TO YOU IN YOUR JURISDICTION, ALTOVA AND ITS SUPPLIERS MAKE NO WARRANTIES, CONDITIONS, REPRESENTATIONS OR TERMS, EXPRESS OR IMPLIED, WHETHER BY STATUTE, COMMON LAW, CUSTOM, USAGE OR OTHERWISE AS TO ANY OTHER MATTERS. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, ALTOVA AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, INFORMATIONAL CONTENT OR ACCURACY, QUIET ENJOYMENT, TITLE AND NON-INFRINGEMENT, WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS, WHICH VARY FROM STATE/JURISDICTION TO STATE/JURISDICTION.

(c) Limitation of Liability. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW EVEN IF A REMEDY FAILS ITS ESSENTIAL PURPOSE, IN NO EVENT SHALL ALTOVA

OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF ALTOVA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, ALTOVA'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS AGREEMENT SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT. Because some states and jurisdictions do not allow the exclusion or limitation of liability, the above limitation may not apply to you. In such states and jurisdictions, Altova's liability shall be limited to the greatest extent permitted by law and the limitations or exclusions of warranties and liability contained herein do not prejudice applicable statutory consumer rights of person acquiring goods otherwise than in the course of business. The disclaimer and limited liability above are fundamental to this Agreement between Altova and you.

(d) Infringement Claims. Altova will indemnify and hold you harmless and will defend or settle any claim, suit or proceeding brought against you by a third party that is based upon a claim that the content contained in the Software infringes a copyright or violates an intellectual or proprietary right protected by United States or European Union law ("Claim"), but only to the extent the Claim arises directly out of the use of the Software and subject to the limitations set forth in Section 5 of this Agreement except as otherwise expressly provided. You must notify Altova in writing of any Claim within ten (10) business days after you first receive notice of the Claim, and you shall provide to Altova at no cost such assistance and cooperation as Altova may reasonably request from time to time in connection with the defense of the Claim. Altova shall have sole control over any Claim (including, without limitation, the selection of counsel and the right to settle on your behalf on any terms Altova deems desirable in the sole exercise of its discretion). You may, at your sole cost, retain separate counsel and participate in the defense or settlement negotiations. Altova shall pay actual damages, costs, and attorney fees awarded against you (or payable by you pursuant to a settlement agreement) in connection with a Claim to the extent such direct damages and costs are not reimbursed to you by insurance or a third party, to an aggregate maximum equal to the purchase price of the Software. If the Software or its use becomes the subject of a Claim or its use is enjoined, or if in the opinion of Altova's legal counsel the Software is likely to become the subject of a Claim, Altova shall attempt to resolve the Claim by using commercially reasonable efforts to modify the Software or obtain a license to continue using the Software. If in the opinion of Altova's legal counsel the Claim, the injunction or potential Claim cannot be resolved through reasonable modification or licensing, Altova, at its own election, may terminate this Agreement without penalty, and will refund to you on a pro rata basis any fees paid in advance by you to Altova. THE FOREGOING CONSTITUTES ALTOVA'S SOLE AND EXCLUSIVE LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT. This indemnity does not apply to situations where the alleged infringement, whether patent or otherwise, is the result of a combination of the Altova software and additional elements supplied by you.

6. SUPPORT AND MAINTENANCE

Altova offers multiple optional "Support & Maintenance Package(s)" ("SMP") for the version of Software product edition that you have licensed, which you may elect to purchase in addition to your Software license. The Support Period, hereinafter defined, covered by such SMP shall be delineated at such time as you elect to purchase a SMP. Your rights with respect to support and maintenance as well as your upgrade eligibility depend on your decision to purchase SMP and the level of SMP that you have purchased:

(a) If you have not purchased SMP, you will receive the Software AS IS and will not receive any maintenance releases or updates. However, Altova, at its option and in its sole discretion on

a case by case basis, may decide to offer maintenance releases to you as a courtesy, but these maintenance releases will not include any new features in excess of the feature set at the time of your purchase of the Software. In addition, Altova will provide free technical support to you for thirty (30) days after the date of your purchase (the "Support Period" for the purposes of this paragraph 6(a), and Altova, in its sole discretion on a case by case basis, may also provide free courtesy technical support during your thirty (30) day evaluation period. Technical support is provided via a Web-based support form only, and there is no guaranteed response time.

(b) If you have purchased SMP, then solely for the duration of its delineated Support Period, **you are eligible to receive the version of the Software edition** that you have licensed and all maintenance releases and updates for that edition that are released during your Support Period. For the duration of your SMP's Support Period, you will also be eligible to receive upgrades to the comparable edition of the next version of the Software that succeeds the Software edition that you have licensed for applicable upgrades released during your Support Period. The specific upgrade edition that you are eligible to receive based on your Support Period is further detailed in the SMP that you have purchased. Software that is introduced as separate product is not included in SMP. Maintenance releases, updates and upgrades may or may not include additional features. In addition, Altova will provide Priority Technical Support to you for the duration of the Support Period. Priority Technical Support is provided via a Web-based support form only and Altova will make commercially reasonable efforts to respond via e-mail to all requests within forty-eight (48) hours during Altova's business hours (MO-FR, 8am UTC – 10pm UTC, Austrian and US holidays excluded) and to make reasonable efforts to provide work-arounds to errors reported in the Software.

During the Support Period you may also report any Software problem or error to Altova. If Altova determines that a reported reproducible material error in the Software exists and significantly impairs the usability and utility of the Software, Altova agrees to use reasonable commercial efforts to correct or provide a usable work-around solution in an upcoming maintenance release or update, which is made available at certain times at Altova's sole discretion.

If Altova, in its discretion, requests written verification of an error or malfunction discovered by you or requests supporting example files that exhibit the Software problem, you shall promptly provide such verification or files, by email, telecopy, or overnight mail, setting forth in reasonable detail the respects in which the Software fails to perform. You shall use reasonable efforts to cooperate in diagnosis or study of errors. Altova may include error corrections in maintenance releases, updates, or new major releases of the Software. Altova is not obligated to fix errors that are immaterial. Immaterial errors are those that do not significantly impact use of the Software as determined by Altova in its sole discretion. Whether or not you have purchased the Support & Maintenance Package, technical support only covers issues or questions resulting directly out of the operation of the Software and Altova will not provide you with generic consultation, assistance, or advice under any circumstances.

Updating Software may require the updating of software not covered by this Agreement before installation. Updates of the operating system and application software not specifically covered by this Agreement are your responsibility and will not be provided by Altova under this Agreement. Altova's obligations under this Section 6 are contingent upon your proper use of the Software and your compliance with the terms and conditions of this Agreement at all times. Altova shall be under no obligation to provide the above technical support if, in Altova's opinion, the Software has failed due to the following conditions: (i) damage caused by the relocation of the Software to another location or CPU; (ii) alterations, modifications or attempts to change the Software without Altova's written approval; (iii) causes external to the Software, such as natural disasters, the failure or fluctuation of electrical power, or computer equipment failure; (iv) your failure to maintain the Software at Altova's specified release level; or (v) use of the Software with other software

without Altova's prior written approval. It will be your sole responsibility to: (i) comply with all Altova-specified operating and troubleshooting procedures and then notify Altova immediately of Software malfunction and provide Altova with complete information thereof; (ii) provide for the security of your confidential information; (iii) establish and maintain backup systems and procedures necessary to reconstruct lost or altered files, data or programs.

7. SOFTWARE ACTIVATION, UPDATES AND LICENSE METERING

(a) License Metering. The Software includes a built-in license metering module that is designed to assist you with monitoring license compliance in small local area networks (LAN). The metering module attempts to communicate with other machines on your local area network (LAN). You permit Altova to use your internal network for license monitoring for this purpose. This license metering module may be used to assist with your license compliance but should not be the sole method. Should your firewall settings block said communications, you must deploy an accurate means of monitoring usage by the end user and preventing users from using the Software more than the Permitted Number.

(b) License Compliance Monitoring. You are required to utilize a process or tool to ensure that the Permitted Number is not exceeded. Without prejudice or waiver of any potential violations of the Agreement, Altova may provide you with additional compliance tools should you be unable to accurately account for license usage within your organization. If provided with such a tool by Altova, you (a) are required to use it in order to comply with the terms of this Agreement and (b) permit Altova to use your internal network for license monitoring and metering and to generate compliance reports that are communicated to Altova from time to time.

(c) Software Activation. **The Software may use your internal network and Internet connection for the purpose of transmitting license-related data at the time of installation, registration, use, or update to an Altova Master License Server and validating the authenticity of the license-related data in order to protect Altova against unlicensed or illegal use of the Software and to improve customer service. Activation is based on the exchange of license related data between your computer and the Altova Master License Server. You agree that Altova may use these measures and you agree to follow any applicable requirements. You further agree that use of license key codes that are not or were not generated by Altova and lawfully obtained from Altova, or an authorized reseller as part of an effort to activate or use the Software violates Altova's intellectual property rights as well as the terms of this Agreement. You agree that efforts to circumvent or disable Altova's copyright protection mechanisms, the license management mechanism, or the Altova Master License Server violate Altova's intellectual property rights as well as the terms of this Agreement. Altova expressly reserves the rights to seek all available legal and equitable remedies to prevent such actions and to recover lost profits, damages and costs.**

(d) LiveUpdate. Altova provides a new LiveUpdate notification service to you, which is free of charge. Altova may use your internal network and Internet connection for the purpose of transmitting license-related data to an Altova-operated LiveUpdate server to validate your license at appropriate intervals and determine if there is any update available for you.

(e) Use of Data. The terms and conditions of the Privacy Policy are set out in full at http://www.altova.com/privacy and are incorporated by reference into this Agreement. By your acceptance of the terms of this Agreement and/or use of the Software, you authorize the collection, use and disclosure of information collected by Altova for the purposes provided for in this Agreement and/or the Privacy Policy. Altova has the right in its sole discretion to amend this provision of the Agreement and/or Privacy Policy at any time. You are encouraged to review the

 © 2015 Altova GmbH

terms of the Privacy Policy as posted on the Altova Web site from time to time.

(f) Audit Rights. You agree that Altova may audit your use of the Software for compliance with the terms of this Agreement at any time, upon reasonable notice. In the event that such audit reveals any use of the Software by you other than in full compliance with the terms of this Agreement, you shall reimburse Altova for all reasonable expenses related to such audit in addition to any other liabilities you may incur as a result of such non-compliance.

(g) Notice to European Users. Please note that the information as described in paragraph 7(d) above may be transferred outside of the European Economic Area, for purposes of processing, analysis, and review, by Altova, Inc., a company located in Beverly, Massachusetts, U.S.A., or its subsidiaries or Altova's subsidiaries or divisions, or authorized partners, located worldwide. You are advised that the United States uses a sectoral model of privacy protection that relies on a mix of legislation, governmental regulation, and self-regulation. You are further advised that the Council of the European Union has found that this model does not provide "adequate" privacy protections as contemplated by Article 25 of the European Union's Data Directive. (Directive 95/46/EC, 1995 O.J. (L 281) 31). Article 26 of the European Union's Data Directive allows for transfer of personal data from the European Union to a third country if the individual has unambiguously given his consent to the transfer of personal information, regardless of the third country's level of protection. By agreeing to this Agreement, you consent to the transfer of all such information to the United States and the processing of that information as described in this Agreement and the Privacy Policy.

8. TERM AND TERMINATION

This Agreement may be terminated (a) by your giving Altova written notice of termination; (b) by Altova, at its option, giving you written notice of termination if you commit a breach of this Agreement and fail to cure such breach within ten (10) days after notice from Altova; or (c) at the request of an authorized Altova reseller in the event that you fail to make your license payment or other monies due and payable. In addition the Agreement governing your use of a previous version of the Software that you have upgraded or updated is terminated upon your acceptance of the terms and conditions of the Agreement accompanying such upgrade or update. Upon any termination of the Agreement, you must cease all use of the Software that this Agreement governs, destroy all copies then in your possession or control and take such other actions as Altova may reasonably request to ensure that no copies of the Software remain in your possession or control. The terms and conditions set forth in Sections 1(h), 1(i), 1(j), 1(k), 1(l), 2, 5, 7, 9, 10, 11, and 11 survive termination as applicable.

9. RESTRICTED RIGHTS NOTICE AND EXPORT RESTRICTIONS

The Software was developed entirely at private expense and is commercial computer software provided with **RESTRICTED RIGHTS**. Use, duplication or disclosure by the U.S. Government or a U.S. Government contractor or subcontractor is subject to the restrictions set forth in this Agreement and as provided in FAR 12.211 and 12.212 (48 C.F.R. §12.211 and 12.212) or DFARS 227. 7202 (48 C.F.R. §227-7202) as applicable. Consistent with the above as applicable, Commercial Computer Software and Commercial Computer Documentation licensed to U.S. government end users only as commercial items and only with those rights as are granted to all other end users under the terms and conditions set forth in this Agreement. Manufacturer is Altova GmbH, Rudolfsplatz 13a/9, A-1010 Vienna, Austria/EU. You may not use or otherwise export or re-export the Software or Documentation except as authorized by United States law and the laws of the jurisdiction in which the Software was obtained. In particular, but without limitation, the Software or Documentation may not be exported or re-exported (i) into (or to a national or resident of) any U.S. embargoed country or (ii) to anyone on the U.S. Treasury Department's list

of Specially Designated Nationals or the U.S. Department of Commerce's Table of Denial Orders. By using the Software, you represent and warrant that you are not located in, under control of, or a national or resident of any such country or on any such list.

10. U.S. GOVERNMENT ENTITIES

Notwithstanding the foregoing, if you are an agency, instrumentality or department of the federal government of the United States, then this Agreement shall be governed in accordance with the laws of the United States of America, and in the absence of applicable federal law, the laws of the Commonwealth of Massachusetts will apply. Further, and notwithstanding anything to the contrary in this Agreement (including but not limited to Section 5 (Indemnification)), all claims, demands, complaints and disputes will be subject to the Contract Disputes Act (41 U.S.C. §§7101 *et seq.*), the Tucker Act (28 U.S.C. §1346(a) and §1491), or the Federal Tort Claims Act (28 U.S.C. §§1346(b), 2401-2402, 2671-2672, 2674-2680), FAR 1.601(a) and 43.102 (Contract Modifications); FAR 12.302(b), as applicable, or other applicable governing authority. For the avoidance of doubt, if you are an agency, instrumentality, or department of the federal, state or local government of the U.S. or a U.S. public and accredited educational institution, then your indemnification obligations are only applicable to the extent they would not cause you to violate any applicable law (e.g., the Anti-Deficiency Act), and you have any legally required authorization or authorizing statute.

11. THIRD PARTY SOFTWARE

The Software may contain third party software which requires notices and/or additional terms and conditions. Such required third party software notices and/or additional terms and conditions are located at our Website at http://www.altova.com/legal_3rdparty.html and are made a part of and incorporated by reference into this Agreement. By accepting this Agreement, you are also accepting the additional terms and conditions, if any, set forth therein.

12. JURISDICTION, CHOICE OF LAW, AND VENUE

If you are located in the European Union and are using the Software in the European Union and not in the United States, then this Agreement will be governed by and construed in accordance with the laws of the Republic of Austria (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the Handelsgericht, Wien (Commercial Court, Vienna) and you further agree and expressly consent to the exercise of personal jurisdiction in the Handelsgericht, Wien (Commercial Court, Vienna) in connection with any such dispute or claim.

If you are located in the United States or are using the Software in the United States then this Agreement will be governed by and construed in accordance with the laws of the Commonwealth of Massachusetts, USA (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the federal or state courts of the Commonwealth of Massachusetts and you further agree and expressly consent to the exercise of personal jurisdiction in the federal or state courts of the Commonwealth of Massachusetts in connection with any such dispute or claim.

If you are located outside of the European Union or the United States and are not using the Software in the United States, then this Agreement will be governed by and construed in accordance with the laws of the Republic of Austria (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree

that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the Handelsgericht, Wien (Commercial Court, Vienna) and you further agree and expressly consent to the exercise of personal jurisdiction in the Handelsgericht Wien (Commercial Court, Vienna) in connection with any such dispute or claim. This Agreement will not be governed by the conflict of law rules of any jurisdiction or the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded.

13. TRANSLATIONS

Where Altova has provided you with a foreign translation of the English language version, you agree that the translation is provided for your convenience only and that the English language version will control. If there is any contradiction between the English language version and a translation, then the English language version shall take precedence.

14. GENERAL PROVISIONS

This Agreement contains the entire agreement and understanding of the parties with respect to the subject matter hereof, and supersedes all prior written and oral understandings of the parties with respect to the subject matter hereof. Any notice or other communication given under this Agreement shall be in writing and shall have been properly given by either of us to the other if sent by certified or registered mail, return receipt requested, or by overnight courier to the address shown on Altova's Web site for Altova and the address shown in Altova's records for you, or such other address as the parties may designate by notice given in the manner set forth above. This Agreement will bind and inure to the benefit of the parties and our respective heirs, personal and legal representatives, affiliates, successors and permitted assigns. The failure of either of us at any time to require performance of any provision hereof shall in no manner affect such party's right at a later time to enforce the same or any other term of this Agreement. This Agreement may be amended only by a document in writing signed by both of us. In the event of a breach or threatened breach of this Agreement by either party, the other shall have all applicable equitable as well as legal remedies. Each party is duly authorized and empowered to enter into and perform this Agreement. If, for any reason, any provision of this Agreement is held invalid or otherwise unenforceable, such invalidity or unenforceability shall not affect the remainder of this Agreement, and this Agreement shall continue in full force and effect to the fullest extent allowed by law. The parties knowingly and expressly consent to the foregoing terms and conditions.

Last updated: 2015/09/03

Chapter 12

Glossary

12 Glossary

The glossary section includes the list of terms pertaining to MapForce.

© 2015 Altova GmbH

12.1 C

Component

In MapForce, the term "component" is what represents visually the structure (schema) of your data, or how data is to be transformed (functions). Components are the central building pieces of any mapping. On the mapping area, components appear as rectangles. The following are examples of MapForce components:

- Constants
- Filters
- Conditions
- Function components
- EDI documents (UN/EDIFACT, ANSI X12, HL7)
- Excel 2007+ files
- Simple input components
- Simple output components
- XML Schemas and DTDs

Connection

A connection is a line that you can draw between two connectors. By drawing connections, you instruct MapForce to transform data in a specific way (for example, read data from an XML document and write it to another XML document).

Connector

A connector is a small triangle displayed on the left or right side of a component. The connectors displayed on the left of a component provide data entry points *to that component*. The connectors displayed on the right of a component provide data exit points *from that component*.

12.2 F

Fixed Length Field (FLF)

A common text format where data is conventionally separated into fields which have a fixed length (for example, the first 5 characters of every row represent a transaction ID, and the next 20 characters represent a transaction description).

FlexText

FlexText is a module in MapForce Enterprise Edition which enables you to convert data from non-standard or legacy text files of high complexity to other formats supported by MapForce, and vice versa.

© 2015 Altova GmbH

12.3 I

Input component

An input component is a MapForce component that enables you to pass simple values to a mapping. Input components are commonly used to pass file names or other string values to a mapping at runtime. Input components should not be confused with source components.

12.4 M

MapForce

MapForce is a Windows-based, multi-purpose IDE (integrated development environment) that enables you to transform data from one format to another, or from one schema to another, by means of a visual, "drag-and-drop" -style graphical user interface that does not require writing any program code. In fact, MapForce generates for you the program code which performs the actual data transformation (or data mapping). When you prefer not to generate program code, you can just run the transformation using the MapForce built-in transformation language (available in the MapForce Professional or Enterprise Editions).

Mapping

A MapForce mapping design (or simply "mapping") is the visual representation of how data is to be transformed from one format to another. A mapping consists of components that you add to the MapForce mapping area in order to create your data transformations (for example, convert XML documents from one schema to another). A valid mapping consists of one or several source components connected to one or several target components. You can run a mapping and preview its result directly in MapForce. You can generate code and execute it externally. You can also compile a mapping to a MapForce execution file and automate mapping execution using MapForce Server or FlowForce Server. MapForce saves mappings as files with .mfd extension.

MFF

The file name extension of MapForce function files.

MFD

The file name extrension of MapForce design documents (mappings).

© 2015 Altova GmbH

12.5 O

Output component

An output component (or "simple ouput") is a MapForce component which enables you to return a string value from the mapping. Output components represent just one possible type of target components, but should not be confused with the latter.

12.6 P

parent-context

parent-context is an optional argument in some MapForce core aggregation functions such as **`min`**, **`max`**, **`avg`**, **`count`**. In a source component which has multiple hierarchical sequences, the parent context determines the set of nodes on which the function should operate.

© 2015 Altova GmbH

12.7 S

Source component

A source component is a component from which MapForce reads data. When you run the mapping, MapForce reads the data supplied by the connector of the source component, converts it to the required type, and sends it to the connector of the target component.

12.8 T

Target component

A target component is a component to which MapForce writes data. When you run the mapping, a target component instructs MapForce to either generate a file (or multiple files) or output the result as a string value for further processing in an external program. A target component is the opposite of a source component.

© 2015 Altova GmbH

Index

.

.NET extension functions,
constructors, 448
datatype conversions, .NET to XPath/XQuery, 452
datatype conversions, XPath/XQuery to .NET, 451
for XSLT and XQuery, 446
instance methods, instance fields, 450
overview, 446
static methods, static fields, 449

A

A to Z,
sort component, 157
About MapForce, 379
abs,
as MapForce function (in xpath2 | numeric functions), 313
Activating the software, 378
Add,
as MapForce function (in core | math functions), 280
global resource file, 330
user-def. functions, 213
Aggregate,
function - using named templates, 254
Altova Engines,
in Altova products, 456
Altova extensions,
chart functions (see chart functions), 390
Altova website, 379
Altova XML Parser,
about, 456
Any,
xs:any, 197
Application workflow,
using global resources, 337
Assign,
global resource to component, 332
ATTLIST,
DTD namespace URIs, 189
auto-number,
as MapForce function (in core | generator functions), 274
avg,
as MapForce function (in core | aggregate functions), 261

B

Background Information, 456
base-uri,
as MapForce function (in xpath2 | accessors library), 306
Bool,
output if false, 227
boolean,
as MapForce function (in core | conversion functions), 265
Builder,
user-defined function, 213
Built-in engine,
definition, 67
using, 67

C

Call,
template, 250
CDATA, 196
ceiling,
as MapForce function (in core | math functions), 280
Chained,
mapping - code generation, 124
Chained mapping,
display final component using Stylevision, 119, 124
Change,
configuration - global resource, 333
char-from-code,
as MapForce function (in core | string functions), 297
Children,
standard with children, 114
Code,
inline functions & code size, 221
code generation,
of chained mappings, 124
Code point,
collation, 157
code-from-char,

code-from-char,
as MapForce function (in core | string functions), 297
Collation, 157
locale collation, 157
sort component, 157
unicode code point, 157
Comments,
Adding to target files, 195
Companion software,
for download, 379
Complex,
function - inline, 221
User-defined complex input, 232
User-defined complex output, 237
User-defined function, 231, 237
Complex type,
sorting, 157
Component,
as application menu, 363
assign global resource, 332
definition of, 477
deleted items, 95
sort data, 157
Component download center,
at Altova web site, 379
Components,
adding to the mapping, 63
aligning, 78
changing settings, 79
overview, 76
processing sequence, 180
searching, 77
Compute once,
variable, 149
Compute when,
variable, 149
concat,
as MapForce function (in core | string functions), 298
Concatenate,
filters - don't, 164
Configuration, 330
add to global resource, 330
copy existing, 330
switch - global resource, 333
Connection,
as application menu, 364
definition of, 477
Connections,
moving to a different component, 90
preserving on root element change, 90
type driven, 115
Connector,
definition of, 477
Connectors,
copy-all, 115
Consolidating data,
merging XML files, 202
Constructor,
XSLT2, 254
contains,
as MapForce function (in core | string functions), 298
Copy all,
mapping method, 107
Copy-all,
and filters, 115
connectors, 115
resolve / delete connectors, 115
Copyright information, 459
count,
as MapForce function (in core | aggregate functions), 261
Create,
user-defined function, 213
current,
as MapForce function (in xslt | xslt functions library), 318
current-date,
as MapForce function (in xpath2 | context functions), 308
current-dateTime,
as MapForce function (in xpath2 | context functions), 308
current-time,
as MapForce function (in xpath2 | context functions), 308
Custom,
XSLT 2.0 functions, 254
XSLT functions, 250

D

Datatype,
explicit - implicit, 254
Date,
XSLT 2.0 constructor, 254
Default,
configuration - global resource, 330
input value, 227
default-collation,

© 2015 Altova GmbH

default-collation,
as MapForce function (in xpath2 | context functions), 308
Definition file,
globalresource.xml, 329
Delete,
copy-all connections, 115
deletions - missing items, 95
user-defined function, 213
Derived types,
mapping to/from, 190
Digital signature,
creating in XML output, 185
distinct-values,
as MapForce function (in core | sequence functions), 285
Distribution,
of Altova's software products, 459, 461
divide,
as MapForce function (in core | math functions), 281
document,
as MapForce function (in xslt | xslt functions library), 318
DoTransform.bat,
execute with RaptorXML Server, 323
DTD,
source and target, 189
Duplicate input, 36
adding, 363

E

Edit,
as application menu, 360
Element,
recursive element in XML Schema, 241
element-available,
as MapForce function (in xslt | xslt functions library), 319
Encoding settings,
in XML output, 185
End User License Agreement, 459, 461
equal,
as MapForce function (in core | logical functions), 277
equal-or-greater,
as MapForce function (in core | logical functions), 277
equal-or-less,
as MapForce function (in core | logical functions), 278
Erase,
delete user-defined func., 213
Evaluation key,
for your Altova software, 378
Evaluation period,
of Altova's software products, 459, 461
Example,
recursive user-defined mapping, 241
exists,
as MapForce function (in core | sequence functions), 286
Explicit,
datatype, 254
Export,
user-defined function, 213
Extension functions for XSLT and XQuery, 436
Extension Functions in .NET for XSLT and XQuery,
see under .NET extension functions, 446
Extension Functions in Java for XSLT and XQuery,
see under Java extension functions, 437
Extension Functions in MSXSL scripts, 452

F

false,
as MapForce function (in xpath2 | boolean functions), 307
FAQs on MapForce, 379
File, 330
add resource configuration, 330
as application menu, 357
as button on a component, 79
as button on components, 131
define global resource, 330
File names,
supplying as mapping input parameters, 135
File paths,
fixing broken references, 104
in generated code, 105
relative versus absolute, 102, 105
File/String,
as button on a component, 79
as button on components, 131
File: (default),
as name of root node, 131
File: <dynamic>,
as name of root node, 131
Filter, 164
component - tips, 164
concatenate - don't, 164

© 2015 Altova GmbH

Filter, 164
copy-all connector, 115
map parent items, 164
merging XML files, 202
priority context, 164
first-items,
as MapForce function (in core | sequence functions), 287
FlexText,
definition of, 478
FLF,
definition of, 478
floor,
as MapForce function (in core | math functions), 281
Folders,
as a global resource, 335
format-date,
as MapForce function (in core | conversion functions), 265
format-dateTime,
as MapForce function (in core | conversion functions), 266
format-number,
as MapForce function (in core | conversion functions), 269
format-time,
as MapForce function (in core | conversion functions), 271
Function, 213, 221, 222
adding custom XSLT, 250
adding custom XSLT 2.0, 254
as application menu, 365
Changing type of user-defined, 213
complex - inline, 221
exporting user-defined, 213
inline, 221
nested user-defined, 227
restrictions in user-defined, 213
standard user-defined function, 222
sum, 254
user-defined, 213
user-defined - changing type, 213
user-defined function, 243
user-defined look-up function, 222
visual builder, 213
function-available,
as MapForce function (in xslt | xslt functions library), 319
Functions,
adding as mapping components, 209
adding parameters to, 209
deleting parameters from, 209
finding in the Libraries window, 209
importing user-defined, 213
viewing the argument data type of, 209
viewing the description of, 209

G

Generate,
code & inline functions, 221
generate-id,
as MapForce function (in xslt | xslt functions library), 319
generate-sequence,
as MapForce function (in core | sequence functions), 287
get-fileext,
as MapForce function (in core | file path functions), 272
get-folder,
as MapForce function (in core | file path functions), 272
Global resource, 330
activate, 333
assign to component, 332
copy configuration, 330
default configuration, 330
folders as, 335
properties, 345
start workflow, 342
workflow, 337
Global resources,
define resource file, 330
definition file, 329
toolbar, 329
Globalresource.xml,
resource definition, 329
greater,
as MapForce function (in core | logical functions), 278
group-adjacent,
as MapForce function (in core | sequence functions), 287
group-by,
as MapForce function (in core | sequence functions), 288
group-ending-with,
as MapForce function (in core | sequence functions), 289
group-into-blocks,
as MapForce function (in core | sequence functions), 289
group-starting-with,
as MapForce function (in core | sequence functions), 289

© 2015 Altova GmbH

H

Health Level 7,
example, 205
Help,
as application menu, 378
see Onscreen Help, 378
HL7 2.6 to 3.x,
example, 205

I

Implicit,
datatype, 254
implicit-timezone,
as MapForce function (in xpath2 | context functions), 308
Import,
user-def. functions, 213
Include,
XSLT, 250
XSLT 2.0, 254
Inline, 213
functions and code size, 221
Inline / Standard,
user-defined functions, 221
Input, 227
default value, 227
optional parameters, 227
Input component,
definition of, 479
Insert,
as application menu, 361
Instance,
changing the path reference to, 102
Intermediate variables, 149
Internet usage,
in Altova products, 457
is-xsi-nil,
as MapForce function (in core | node functions), 283
Item,
missing, 95
item-at,
as MapForce function (in core | sequence functions), 290
items-from-till,
as MapForce function (in core | sequence functions), 290

J

Java extension functions,
constructors, 442
datatype conversions, Java to Xpath/XQuery, 446
datatype conversions, XPath/XQuery to Java, 444
for XSLT and XQuery, 437
instance methods, instance fields, 444
overview, 437
static methods, static fields, 443
user-defined class files, 438
user-defined JAR files, 441

K

Keeping data,
when using value-map, 169
Keeping data unchanged,
passing through a value-map, 169
Key,
sort key, 157
Key-codes,
for your Altova software, 378

L

last,
as MapForce function (in xpath2 | context functions), 308
last-items,
as MapForce function (in core | sequence functions), 291
Legal information, 459
less,
as MapForce function (in core | logical functions), 278
Libraries,
and user-defined functions, 213
Libraries window,
finding functions in, 209
Library,
adding XSLT 2.0 functions, 254
adding XSLT functions, 250

© 2015 Altova GmbH

Library,
import user-def. functions, 213
License, 461
information about, 459
License metering,
in Altova products, 460
Licenses,
for your Altova software, 378
Locale collation, 157
logical-and,
as MapForce function (in core | logical functions), 278
logical-not,
as MapForce function (in core | logical functions), 279
logical-or,
as MapForce function (in core | logical functions), 279
Lookup table,
properties, 171
value map table, 166

M

main-mfd-filepath,
as MapForce function (in core | file path functions), 272
MapForce,
basic concepts, 15
overview, 9
MapForce samples,
location on disk, 21
Mapping, 114
creating, 63
definition of, 480
processing sequence, 180
source driven - mixed content, 107
standard mapping, 114
target driven, 114
type driven, 115
validating, 68
Mapping input,
supplying custom file name as, 135
Supplying multiple files as, 131, 133, 134
Mapping methods,
standard, 107
standard / mixed / copy all, 107
target-driven, 107
Mapping output,
Generating multiple files as, 131, 134
Marked items,
missing items, 95
max,
as MapForce function (in core | aggregate functions), 262
max-string,
as MapForce function (in core | aggregate functions), 262
Memory requirements, 456
Merging,
XML files, 202
mfd,
as file extension, 480
mfd-filepath,
as MapForce function (in core | file path functions), 273
mff,
and user-defined functions, 213
as file extension, 480
mfp,
as file extension, 480
mft,
as file extrension, 480
Microsoft SharePoint Server,
adding files as components from, 64
min,
as MapForce function (in core | aggregate functions), 263
min-string,
as MapForce function (in core | aggregate functions), 263
Missing items, 95
Mixed, 107
content mapping, 107
content mapping example, 112
content mapping method, 107
source-driven mapping, 107
standard mapping, 114
modulus,
as MapForce function (in core | math functions), 281
msxsl:script, 452
Multiple source,
to single target, 202
multiply,
as MapForce function (in core | math functions), 282

N

Named,
template - namespaces, 250
Named template,

© 2015 Altova GmbH

Named template,
summing nodes, 254
Namespace,
named template, 250
Namespace URI,
DTD, 189
Namespace URIs,
and QNames, 192
Namespaces,
and wildcards (xs:any), 197
Nested,
user-defined functions, 227
nillable,
as attribute in XML schema, 192
Node,
summing multiple, 254
node-name,
as MapForce function (in core | node functions), 283
as MapForce function (in xpath2 | accessors library), 306
normalize-space,
as MapForce function (in core | string functions), 298
not-equal,
as MapForce function (in core | logical functions), 279
not-exists,
as MapForce function (in core | sequence functions), 291
number,
as MapForce function (in core | conversion functions), 271

O

Onscreen help,
index of, 378
searching, 378
table of contents, 378
Optional,
input parameters, 227
Order,
components are processed, 180
Ordering Altova software, 378
Ordering data,
sort component, 157
OS,
for Altova products, 456
Output, 227
as application menu, 366
parameter, 227
previewing, 70
saving, 70
user-defined if bool = false, 227
validating, 69
Output component,
definition of, 481

P

Parameter, 227
optional, 227
output, 227
Parent,
mapping and filters, 164
Parent context,
variable, 149
parent-context,
definition of, 482
Parser,
built into Altova products, 456
Passing through data,
unchanged through value-map, 169
Paths in generated code,
making absolute, 74
Platforms,
for Altova products, 456
position,
as MapForce function (in core | sequence functions), 292
Priority,
and filters, 164
Priority Context,
setting on functions, 211
Processing Instructions,
Adding to target files, 195
Processing Instructions and Comments,
mapping, 108
Processing sequence,
of components in a mapping, 180
Processors,
for download, 379
Properties,
value map table, 171

© 2015 Altova GmbH

Q

QName support, 192
Question mark,
missing items, 95

R

RaptorXML Server,
executing a transformation, 323
Recursive,
calls in functions, 221
user-defined function, 243
user-defined mapping, 241
Reference, 356
Registering your Altova software, 378
Regular expressions,
as parameter to the "match-pattern" function, 257
as parameter to the "tokenize-regexp" function, 257
Remove,
copy-all connections, 115
remove-fileext,
as MapForce function (in core | file path functions), 273
remove-folder,
as MapForce function (in core | file path functions), 273
replace-fileext,
as MapForce function (in core | file path functions), 273
replicate-item,
as MapForce function (in core | sequence functions), 295
replicate-sequence,
as MapForce function (in core | sequence functions), 296
resolve-filepath,
as MapForce function (in core | file path functions), 274
resolve-uri,
as MapForce function (in xpath2 | anyURI functions), 306
Resource,
folder, 335
global resource properties, 345
Result of Transformation,
global resources, 337
Retaining data,
passing through vlaue-map, 169
Root,
element of target, 202
round,
as MapForce function (in core | math functions), 282
round-half-to-even,
as MapForce function (in xpath2 | numeric functions), 313
round-precision,
as MapForce function (in core | math functions), 282

S

Schema,
and XML mapping, 185
changing the path reference to, 102
generating for an XML file, 185
recursive elements, 241
Scripts in XSLT/XQuery,
see under Extension functions, 436
Search,
functions in the Libraries window, 209
items within mapping components, 77
Section,
CDATA, 196
Sequence,
of processing components, 180
set-empty,
as MapForce function (in core | sequence functions), 297
set-xsi-nil,
as MapForce function (in core | node functions), 284
Simple type,
sorting, 157
Single target,
multiple sources, 202
skip-first-items,
as MapForce function (in core | sequence functions), 297
Software product license, 461
Sort,
sort component, 157
Sort key,
sort component, 157
Sort order,
changing, 157
Source component,
definition of, 483
Source-driven,
- mixed content mapping, 107
vs. standard mapping, 114

© 2015 Altova GmbH

SQLite,
changing database path to absolute in generated code, 105
Standard, 114
mapping method, 107
mapping with children, 114
mixed content mapping, 114
user-defined function, 222
vs source-driven mapping, 114
starts-with,
as MapForce function (in core | string functions), 299
static-node-annotation,
as MapForce function (in core | node functions), 284
static-node-name,
as MapForce function (in core | node functions), 285
string,
as MapForce function (in core | conversion functions), 272
as MapForce function (in xpath2 | accessors library), 306
string-join,
as MapForce function (in core | aggregate functions), 264
string-length,
as MapForce function (in core | string functions), 299
Stylevision,
chained mapping - final component, 119, 124
substitute-missing,
as MapForce function (in core | sequence functions), 297
substitute-missing-with-xsi-nil,
as MapForce function (in core | node functions), 285
substring,
as MapForce function (in core | string functions), 299
substring-after,
as MapForce function (in core | string functions), 299
substring-before,
as MapForce function (in core | string functions), 300
subtract,
as MapForce function (in core | math functions), 282
sum,
as MapForce function (in core | aggregate functions), 264
nodes in XSLT 1.0, 254
Support for MapForce, 379
Switch,
configuration - global resource, 333
system-property,
as MapForce function (in xslt | xslt functions library), 320

T

Table,
lookup - value map, 166
Table data,
sorting, 157
Target,
root element, 202
Target component,
definition of, 484
Target-driven,
mapping, 114
Target-driven mapping, 107
Technical Information, 456
Technical support for MapForce, 379
Template,
calling, 250
named - summing, 254
tokenize,
as MapForce function (in core | string functions), 300
tokenize-by-length,
as MapForce function (in core | string functions), 302
tokenize-regexp,
as MapForce function (in core | string functions), 304
Toolbar,
global resource, 329
Tools,
as application menu, 370
Transform,
input data - value map, 166
Transformation language,
selecting, 67
Transformations,
RaptorXML Server, 323
translate (in core | string functions),
as MapForce function, 305
true,
as MapForce function (in xpath2 | boolean functions), 307
Type driven,
connections, 115
Types,
derived types - xsi:type, 190

© 2015 Altova GmbH

U

Unicode,
code point collation, 157
Unicode support,
in Altova products, 457
unparsed-entity-uri,
as MapForce function (in xslt | xslt functions library), 320
URI,
in DTDs, 189
URIs,
and QNames, 192
URL,
adding files as components from, 64
User defined, 213, 222, 227
changing function type, 213
complex input, 232
complex output, 237
deleting, 213
function - inline / standard, 221
function - standard, 222
functions, 213
functions - complex, 231, 237
functions - restrictions, 213
functions changing type of, 213
importing/exporting, 213
look-up functions, 222
nested functions, 227
output if bool = false, 227
User manual,
see also Onscreen Help, 378
User-defined, 213
functions, 213
functions & mffs, 213
user-defined function,
recursive, 243
Using,
global resources, 333

V

Validate,
mapping design, 68
mapping output, 69
Validator,
in Altova products, 456
Value,
default, 227
Value-Map,
lookup table, 166
lookup table - properties, 171
passing data unchanged, 169
Variable,
inserting, 149
intermediate variable, 149
use cases, 149
View,
as application menu, 368
Visual function builder, 213

W

WebDAV Server,
adding files as components from, 64
Wildcards,
xs:any - xs:anyAtrribute, 197
Windows,
support for Altova products, 456
Workflow,
start - global resource, 342
using global resource, 337

X

XML declaration,
suppressing from output, 185
XML files,
generate from single XML source, 136
XML output,
changing enconding settings, 185
changing instance file name, 185
changing schema, 185
creating digital signature, 185
XML Parser,
about, 456
XML to XML, 185
XPath,

© 2015 Altova GmbH

XPath,
summing multiple nodes, 254
XQuery,
Extension functions, 436
XQuery processor,
in Altova products, 456
xs: any (xs:anyAttribute), 197
xsi:nil,
as attribute in XML instance, 192
xsi:type,
mapping to derived types, 190
XSLT, 250
adding custom functions, 250
Extension functions, 436
previewing the generated code, 71
template namespace, 250
XSLT 2.0,
adding custom functions, 254
XSLT processors,
in Altova products, 456
XSLT2.0,
date constructor, 254

Z

Z to A,
sort component, 157

© 2015 Altova GmbH